A COMPUTATIONAL PLATFORM FOR ANALYZING THE C. ELEGANS NERVOUS SYSTEM

A University Thesis Presented to the Faculty

of

California State University, East Bay, and Stanford University

Thesis Work was Performed Entirely at Stanford University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computational Biology

By

Eric W. Wasiolek

September, 2007

A COMPUTATIONAL PLATFORM FOR ANALYZING THE C. ELEGANS NERVOUS SYSTEM

By

Eric W. Wasiolek

i.

ABSTRACT

A computer program was developed to combine developmental, neural, and genetic information to predict the neural and synaptic development of *C. elegans*. The program divides development into development stages based upon the number of cell divisions required to generate each neuron. A representation of the synaptic network for each development stage is generated and this represents completely novel information, although the veracity of this representation awaits experimental verification. Further novel computational tools were developed to analyze the synaptic network of the developing animal. These tools find pathways between an input set of sensory neurons and output set of motor neurons to indicate circuit formation and the genetic makeup of circuit formation. Other tools analyze the number of inputs and outputs through neurons or neuron subpaths indicating high throughput neurons and ganglia worthy of further biological analysis. Finally, a relationship between development and synapse formation was sought by correlating lineage distance and synaptic distance.

A COMPUTATIONAL PLATFORM FOR ANALYZING

THE C. ELEGANS NERVOUS SYSTEM

By

Eric W. Wasiolek

Approved:	Date:
Dr. Eric Suess – Chair – Statistics	6/9/08
Dr. Michael Hedrick – Neuroscience	6/9/08
Dr. Jeff Shrager – Stanford	6/9/08
Dr. Maria Gallegos – Developmental Biology	6/9/08
Dr. Jim Daley – Computer Science	6/9/08
Dr. Claudia Stone – Molecular Biology	2008 June 11

ACKNOWLEDGEMENTS

Dr. Jeff Shrager, professor of Symbolic Systems at Stanford University was key to this project. Dr. Shrager provided feedback and direction on both the computational and biological aspects of the project, as well as provided the computer facilities at Stanford to develop and test the programs. Hannah Teichmann, a Phd student in neuroscience at Stanford provided valuable feedback on the biology of *C. elegans*. The graph data structure and some graph algorithms were provided by William Ford and William Topp of the University of the Pacific Computer Science Department for some of the applications here developed. Other thanks go to the advisorial professors at Cal State East Bay, including Dr. Hedrick (Neuroscience), Dr. Suess (Statistics), Dr. Daley (Computer Science), Dr. Holz (Computer Science), Dr. Stone (Molecular Biology), and Dr. Gallegos (Developmental Biology) who provided feedback and review of the materials.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EQUATIONS AND CORRELATIONS

INTRODUCTION

This project involved building a system of programs which jointly allow analysis of the *C. elegans* nervous system (and potentially other nervous systems), by combining biological information from diverse sources to generate novel biological information which is not available in any existing data sources (WormBase, WormBook, or WormAtlas). Specifically, three types of analysis were performed. First, a developmental representation of the *C. elegans* nervous system was formed by combining developmental, genetic, and synaptic network information. Notably, a representation of the developing synaptic network was generated, which represents completely novel information. Secondly, various synaptic pathway analyses were performed by programs which analyze a representation of the synaptic network of *C. elegans*. Finally, programs were used to predict synaptic formation from developmental information. *C. elegans* was chosen as the model organism because of the completeness of information on the *C. elegans* nervous system, where all nerve cells and synapses have been identified. These nerve cells and synapses are represented in database accessible files (the nerve cell by a string indicating the nerve cell designator, and the synapse by a line in a file which contains the two nerve cell designators participating in the synapse). In addition, the complete lineage of nerve cells from the zygote is known for *C. elegans*.

Two aspects of this project are discussed in this text: the biology of the *C. elegans* nervous system, including the genetics of neurogenesis, and the computer scientific approach to representing the *C. elegans* nervous system and its development.

Chapter 1

LITERATURE REVIEW

This study is built upon papers previously published by White et al. (1986), and Chklovskii et al. (2006) on the structure of the *C. elegans* nervous system. In this key work, White et al. (1986) delineated all of the 302 neurons and all of the synapses that comprise the *C. elegans* nervous system, and established neuron naming conventions. The work is to date the basic reference for the *C. elegans* nervous system. The publication refers to spreadsheet files which describe the complete wiring diagram of *C. elegans*. Subsequent work identified neurons and synapses in the posterior portion of the worm. Inconsistencies in the wiring diagram were corrected in further works (Hobart and Hall, 1999; Achacosco and Yamamoto, 1992).

While the neurons and synapses were identified by White (1986), Sulston and Horovitz (1977) published an important paper identifying neuronal cell lineages in the developing *C. elegans* upon which this work is also based. Sultson published an updated version of the article with White (1983) in which several additions and modifications to the cell lineage diagram were added. WormBase derives its lineage diagrams from these works done by Sulston et al., which formed the basis of our developmental model. Chklovskii et al., (2006) later calculated lineage distances, as we did, where the number of cell divisions after which two cells development paths diverge is noted and stored as the lineage distance (data is available at: http://www.wormatlas.org/handbook/nshandbook.htm/nswiring.htm).

The studies above and this project use the mathematical techniques of graph theory. The term ‘graph’ means a toplogy in which nodes (points) are connected by edges (lines). The

simplest example of a graph is a map where cities represent nodes (points on the map) and lines connecting cities represent flights. Flights don't exist from every city to every other city (i.e, the graph isn't fully interconnected), and so the graph has a certain topology by virtue of which points are connected by which lines. If there is no line from point A to point D, for example, there is the consideration of the shortest path, i.e., which flights will get someone from point A to point D in the most efficacious manner. Graphs in this study are simply representing neurons (the nodes or points) interconnected by synapses (the edges or lines). In this way a graph may represent a synaptic network of neurons.

Graph theory was eventually adopted by and applied to computer science in the 1960's and 70's resulting graph data structures, called 'networks,' (Even et al. 1975; Ford et al., 1972; Itai et al., 1974; Korfhage et al., 1974; Monroe et al., 1971). At this time, various publications in the computer science community applied graph theory to discrete data structures, resulting in two representation of graphs: adjacency matrices and edge sets. An 'adjacency matrix' is a matrix wherein a bit is set for each vertex that is connected to another vertex, and an 'edge set' is a set of vertices with a set of edges to which the vertex connects for each vertex. The research in this paper uses both graph representations.

Wiring Minimization Influences Synaptic Network Formation

While the neurons, wiring diagram (synapses), and neuronal lineage for *C. elegans* is known and well established, few computational studies have been done to date. Chklovski et al. (2006) however, did a recent computational study of *C. elegans* producing two noteworthy data sets accompany this paper: a connectivity matrix which lists all synapses, including electrical

junctions and neuromuscular junctions, by listing each neuron, along with each neuron it forms a synapse with, and the number of synapses between the neuron pair and a synaptic connectivity matrix. Synaptic connectivity occurs in such a way to minimize wiring costs given neuronal placement. The authors develop a cost minimization function to predict optimal wiring layouts. It is assumed that the cost of wiring the ith and jth neurons is proportional to some power gamma of the distance between them, expressed in this function:

$$C^{\text{int}} = \frac{1}{2\alpha} \sum_i \sum_j A_{ij} |x_i - x_j|^{\xi}, \quad (1)$$

where xi and xj are the neuron positions, and Aij is an element in the adjacency matrix representing the total number of synapses between i and j. Here, Chklovskii et al. are using an idea similar to our own: synaptic distance to measure the distance between neurons and representing these in a synaptic distance matrix. Initially Chkovskii is setting gamma to 2, assuming that the cost of synaptic distance varies as a square of the distance between the neurons. Positions of neurons predicted by the wiring cost minimization function are compared to the actual positions of neurons in *C. elegans*. The result is given in the figure below:

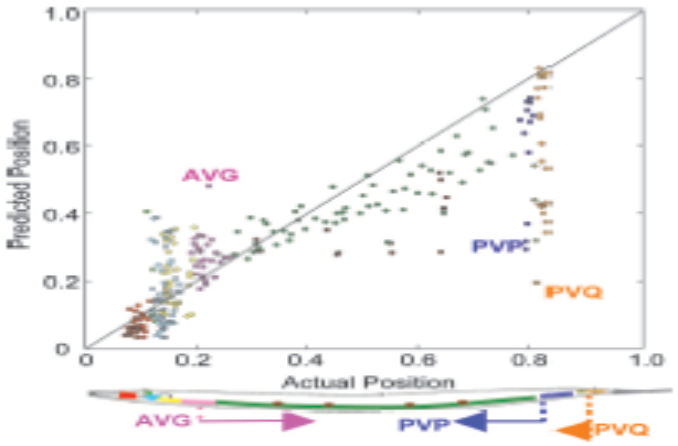

Fig. 2. Neuron positions predicted by the quadratic dedicated-wire model vs. actual neuron positions. (*Upper*) Positions are normalized by the worm body length (0 = head; 1 = tail). Perfect predictions fall on the diagonal. Circles of the same color represent cell bodies belonging to the same ganglion. Three classes of pioneer neurons are labeled. (*Lower*) Schematics depicting the progression of pioneer neurons during worm development. Arrows indicate direction of neurite growth.

As can be seen, there is a reasonably high correspondence between actual positions and predicted positions, along the diagonal line. Still, outliers need to be explained. The authors suggest that biological factors relating to axon guidance, neuronal migration, and command neuron function contribute to non-optimal placements.

Gene Sets Influencing Synapse Formation

A second computational study on the nervous system of *C. elegans* was done by Varadan et al. (2006). Using a systems biology type approach, a technique is used called 'entropy minimization and boolean parsimony' to identify sets of synergistically interacting genes whose joint expression predicts neural connectivity (synapse formation). The authors created an adjacency matrix to describe the connectivity of the mature worm, based upon the published

wiring diagram. The authors worked with the same 280 neurons of the adult hermaphrodite as did our own study. The adjacency matrix is therefore a 280 x 280 binary matrix, where the bit is set if a connection exists between two neurons and clear if it does not. The authors created a second matrix, the expression matrix, where one axis is a set of 292 genes known to be active in *C. elegans* neurons, and the other axis is the 280 neurons. The expression matrix is binary, and a set bit indicates that the gene is active in the neuron, and clear bit indicates the gene is not active in the neuron. The gene expression list is derived from the WormBase browser and represents the same, approximately 300 genes used in this study.

Table 1. Example of a state-count table.

mig-1 (pre)	*unc-8* (post)	*glr-1* (post)	N_0	N_1	Q/Q_{null}
0	0	0	30472	923	1.05
0	0	1	4412	268	2.05
0	1	0	15334	266	0.61
0	1	1	2491	434	5.30
1	0	0	13641	44	0.11
1	0	1	2031	9	0.16
1	1	0	6571	229	1.20
1	1	1	1254	21	0.59

The authors create a state-count table where (this is here a small excerpt) for each neuron pair (of which there are 78,400, or 280 x 280) it is calculated whether a particular permutation of the 292 genes contributes to a synapse or not: if it contributes to a synapse, N1 is incremented, if it does not No is incremented. Note, then, that the sum of No and N1 is 78,400, which is also equal to K^2 below, covering all combinations of the 280 neurons.

$$P(S) = \frac{N_0(S) + N_1(S)}{K^2}$$

$$Q(S) = \frac{N_1(S)}{N_0(S) + N_1(S)} \text{ if } P(S) > 0$$

(2) Gene-synapse association equations

For each gene expression state S, the authors count the number of times that gene is associated with a synapse (associated with a pair of neurons that synapse), N1(S) versus the number of times that gene is not associated with a synapse (No(S)). The two quantities are summed and divided by the number of neurons squared (280^2) to yield P(S), where the probability is equal to a relative frequency, so that the presence of a synapse and the gene expression states are random variables. The latter formula is the probability of a synapse given state S. In the table above, this value is divided by Qnull to determine the likelihood of this combination of genes yielding a synapse. Qnull is determined by dividing the number of chemical synapses, 2195, which incidentally is almost exactly the number in this research project (2199), by K^2 (the number of neuron pairs, 78,400). Therefore Qnull is 0.028. If Q(S) is greater than 0.028, Q(S)/Qnull is greater than one, and this combination of genes probably results in a synapse. If Q(S)/Qnull is less than one, this combination of genes probably does not result in a synapse.

If we know the expression state X of a particular ordered pair of neurons, then the uncertainty of whether a synapse exists is measured by the entropy function H(Q(S)), where H is defined by:

(3) Entropy Function $H(q) = -q\log_2(q) - (1-q)\log_2(1-q).$

The overall uncertainty of whether a synapse is formed is given by the conditional entropy:

(4) Conditional Entropy Function $\sum P(S)H(Q(S))$

where summation is over all S states.

Varadan et al. (2006) used a second technique known as Boolean parsimony to predict synapse formation. By labeling states where the relative frequency is higher than Qnull as one (a set bit) and states with relative frequency lower than Qnull as zero (clear bit), a Karnaugh map may be used to identify a simple Boolean function under which the phenotype (here synapse formation) is present. By rules of Boolean algebra, the simplest Boolean expression describing the combination of expressed genes that result in a synapse may be formed.

In this way the authors were able to identify sets of genes which synergistically imply synapse formation, these sets being available for further experimental investigation. This study distinguishes itself from most experimental work on synapse formation where a single gene is studied that is implicated in some aspect of synaptogenesis (see below).

The Search for Computational Modules in the *C. elegans* Nervous System

Chklovskii et al. (2004) have attempted to identify computational modules within the *C. elegans* nervous system, that is, small sets of neurons interconnected in specific ways that are over-represented in the nervous system and hence have a supposed stereotypic function, much as circuit elements like amplifiers, memory registers, and shifters, are repeated units composed of

specific arrangements of AND, OR, and NOT logic elements, or of specific sets of transistors, resistors, and capacitors.

Chklovski et al. (2004) first noticed that reciprocally connected doublets of neurons were over represented in the *C. elegans* nervous system as opposed to the number that were found in randomly generated networks. This was found to be the case among chemical synapses. Among doublets, only three possible connections exist: A synapses on B, B synapses on A, or A synapses on B and B synapses on A. It was this latter case that was over represented. Secondly, Chklovski et al. investigated neuron triplets, looking for over represented motifs. Triplets were categorized into 13 classes, as indicated in 4 to 15 below:

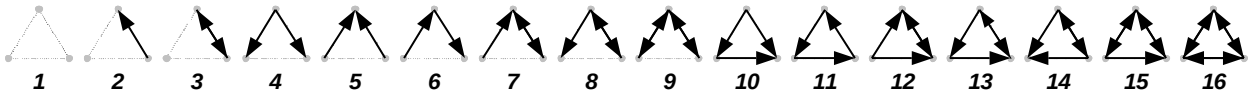

Figure 2. Classes of interconnected neuron triplets.

Among triplets, particularly over-represented compared to random networks was 10, 12, 14, 15, and also 16. One of the most consistently over-represented motifs was #10, the feedforward loop. Here A synapses on C, and A also synapses on B which synapses on C. In this way A sends information to C via two paths, one direct and one indirect.

Quadruplets were analyzed and categorized into 218 distinct classes. Particularly over-represented were class #45 and class #51.

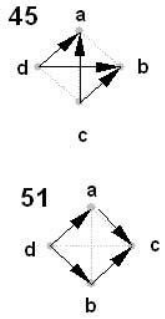

Figure 3. Over-representation of Motifs #45 and #51.

Quintuplets were analyzed but no over-representation of a quintuplet configuration was observed. Some of the results such as the over-representation of the feedforward loop and feedforward quadruplet have been reported previously by White (1986) Hall (1991), and Milo et al. (2002). These over-represented motifs are candidates for computational modules that may perform stereotypical functions and are worthy of further investigation.

Other *C. elegans* Computational Studies

Other computational studies of *C. elegans* have been performed, but not of the nervous system. This is somewhat surprising as *C. elegans* is the only model organism for which we have the entire wiring diagram of the nervous system, making it quite susceptible to computational analysis. These other computational studies involve modeling of vulval development, Sun and Hong (2007) and also Fisher et al. (2005), simulation of cleavage, Kajita et al. (2003), visualization of *C. elegans* MRNA translation and expression, Trutschl et al. (2005), and more traditional bioinformatics applications such as the identification of exons,

LeParc et al. (2007), and computer modeling of regulatory elements and regulatory networks, Thakurta et al., (2002). As will be discussed below, much of the work involving the *C. elegans* synaptic network consists of genetic studies where the effect of a specific gene on synapse formation is considered.

Genetic and Developmental Genetic Studies in Neurogenesis and Synaptogenesis.

These works indicate genes that are involved in various aspects of neurogenesis, axon guidance, and neurotransmission. There is no attempt, in any of these articles, to generate a representation of the synaptic network at various stages of development. Prominent researchers working in the area of *C. elegans* neurogenesis and synaptogenesis include Yishi Jin, Kang Shen of Stanford, Nonet, Bargmann, and Mei Zhen. Yishi Jin has published developmental genetic articles on genes affecting synaptic transmission (2007), genes encoding receptors that function in axon guidance and synapse formation(2006), genes encoding proteins that regulate synaptic vescicle localization (2005), genes regulating motor axon guidance (2003), GABA-ergic neuronal differentiation (2004), and motor neuron fate specification (2000). Kang Shen of Stanford has written extensively on synaptic formation in *C. elegans*, including work on the effective of UNC-6 netrin signaling in synaptogenesis (2007), and selective synapse elimination by E3 ubiquitin ligase (2007). Mike Nonet has conducted several genetic studies on synaptic vesicles formation and function and effect on neurotransmission including the regulation of synaptic transmission by RA3 and RA27m synaptogyrin localization (2003), genes that affect the protein composition of synaptic vescicles, and synaptobrevin mutants and their effect on synaptic transmission (2005). Cornelia Bargmann has published extensively on genes involved in axon

guidance (1999), neuron subtype identity (2002), and chemosensation and mechanosensation (1997). Mei Zhen has investigated genes involved in synaptogenesis and synaptic transmission (2005), and neuronal polarity (2007).

Chapter 2

DEVELOPMENTAL MODEL

Introduction

This study introduces a novel computational model to describe neural development in the genetically tractable model organism *C. elegans*. Developmental stages are defined by the number of cell divisions, and address which neurons are born, which genes are active in those neurons and which synaptic connections are formed by neurons born at each development stage. Furthermore, these data are embedded into an understanding of sensory-motor circuit formation.

A comprehensive description of neuronal connectivity has been provided by electron micrographs reconstructions by White (1986). While this makes *C. elegans* the only organism for which there is a complete wiring diagram in the adult, and recently a number of genes associated with synaptic development have been described, an understanding of how this synaptic development arises over time is lacking. An attempt to approximate the course of synaptic and circuit development in the nervous system is performed by analyzing the cell divisions which give rise to groups of neurons and their gene expression.

Methods

Developmental stages are based upon the number of cell divisions, with neurons born at the same number of divisions belonging to the same stage. Genes associated with each stage is the gene expression list for neurons belonging to the same developmental stage (gene expression data from wormbase, see below). The list of genes associated with each stage is a union of the lists of genes expressed for each neuron that is a part of that stage. The following restrictions

have been used in generating the synaptic network per stage: only synapses where both neurons in the synapse are part of the list of neurons existing by that developmental stage are listed as synapses at that development stage. The analysis of the data set generated has the caveat that synapse formation is not simultaneous with neurogenesis and that synapse formation may take place with different time lags depending on the time occupied by neural migration and axon pathfinding.

A developmental representation of the *C. elegans* nervous system was computed by programmatically relating three information sources: neurogenesis as described by cell lineage, Sulston (1983), gene expression data for each neuron (Wormbase), and a representation of the entire synaptic network of *C. elegans*, White (1986), Chen (2006). The cell lineage indicates the path of development of a specific neuron, given a unique 3-5 character label. For example, the neuron ADAL has the lineage string "AB.plapaaaapp". The lineage string indicates the path of development of a specific neuron. Hence "AB" indicates that ADAL is a descendent of the AB cell, which forms after the FIRST cell division of the zygote. The AB cell then divides into a posterior cell ('p' for 'posterior') and an anterior cell ('a' for 'anterior'). ADAL is a descendent of the posterior cell, as indicated by the 'p' which follows 'AB' in the lineage string. This posterior AB cell then divides into left and right progenitors. ADAL is a descendent of the left cell as indicated by the 'l' which follows "AB.p." In this way the entire lineage string indicates the developmental cell division decisions which give rise to ADAL after 11 divisions. Some strings are short, such as the sensory neuron ADFL with the lineage string "AB.alpppppaa" indicating fewer cell divisions (never fewer than 11) were required to generate the mature neuron, and some are long, such as PVDR's "ABprapapaaTRpaapa" which indicates more cell

divisions were required to generate PVDR. No neurons are born with fewer than 11 cell divisions. Cells with same length lineage strings are hence generated from the same number of divisions, and cells generated with the same number of cell divisions belong to the same development stage. The program 'develstages()' groups neurons into 6 stages, as lineage string lengths increase with the number of cell divisions which range from 11 to 16.

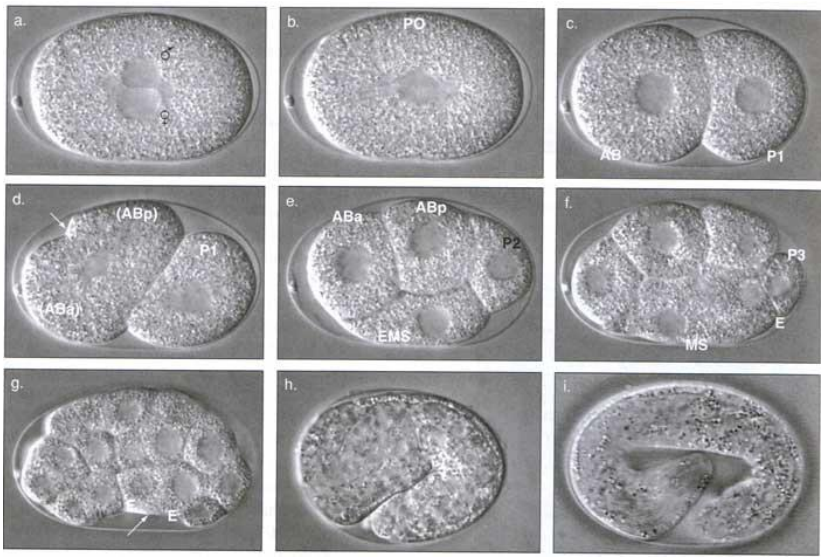

Figure 4. Cleavage of *C. elegans* zygote. The progenitor cell P0 divides into an AB cell and P1 cell. AB then divides into ABa and Abp.

Once the neurons belonging to a specific development stage are known, the synapses to and from those neurons are calculated. A synapse is described by a 'from-neuron' (pre-synapse) and a 'to-neuron' (post-synapse), for example, an ADAL to DB1 synapse is denoted by "ADAL DB1 1" where ADAL and DB1 are neurons and 1 indicates the 'weight' of the synapse (1 for each synapse in this analysis). All synapses corresponding to a stage are found by the program parsing the list of all synapses and picking out those for which both neurons belong to that development stage. A principle used is that synapses cannot form with neurons that are not yet

born. This simple algorithm allows generating the synaptic network which will result from neurons at each stage. Moreover, once development graphs (representations of the synaptic network) for each development stage are generated, they may be fed back into any of the synaptic analysis programs (see Part 2: Synaptic Pathway Analysis) developed in this study.

II. Biological Data Sets

Sources of biological information used in this study will now be discussed.

Neuron List

A list of neurons in the adult hermaphrodite for which synaptic connectivity data exists was used, Chen (2006). Male-specific neurons were excluded to provide a neutral worm model. The list was standardized so that in all cases where this list was referred to and used by one of the programs in this study, it consisted of the same 280 neurons, in the same order. This consistency was required by programs such as the correlation of lineage and synaptic distance (see chapter 4).

Lineage String List

Next, lineage strings corresponding to each neuron were found in WormBase at the address: http://www.wormatlas.org/neuroimageinterface.htm . This is part of the WormAtlas site (see Figure 7).

NEURON	LINEAGE	DESCRIPTION
ADAL	AB.plapaaaapp	Ring interneurons
ADAR	AB.prapaaaapp	Ring interneuron
ADEL	AB.plapaaaapa	Anterior deirid, sensory neuron
ADER	AB.prapaaaapa	Anterior deirid, sensory neuron
ADFL	AB.alpppppaa	Amphid neuron
ADFR	AB.praaappaa	Amphid neuron
ADLL	AB.alpppaad	Amphid neuron
ADLR	AB.praaapaad	Amphid neuron
AFDL	AB.alpppapav	Amphid finger cell
AFDR	AB.praaaapav	Amphid finger cell
AIAL	AB.plppaappa	Amphid interneuron
AIAR	AB.prppaappa	Amphid interneuron
AIBL	AB.plaapappa	Amphid interneuron
AIBR	AB.praapappa	Amphid interneuron
AIML	AB.plpaapppa	Ring interneuron

Figure 5. Lineage Strings. The source of the neuronal lineage string information for *C. elegans* is from the Worm Atlas at: http://www.wormatlas.org/neuroimageinterface.htm . This information is also available in the Pedigree browser of WormBase.

Since the model in this study is based upon 280 cells of the hermaphrodite (the male-specific neurons are excluded), the lineage list, which is the lineage input file, consists of 280 lines. The lineage input file indicates the number of neurons, followed by the name of the neuron and its lineage string on each line. The file was carefully crafted so that the number of characters in the lineage string consistently reflected the number of cell divisions, as this was critical for the development stage algorithm (please refer to Figure 7).

List of Synapses

The information comprising the synaptic network of the mature worm was taken from the website: http://www.ee.columbia.edu/~anastas/ismb2006/, Chen (2006). Three files were used from this site: Post_Synaptic_Neuronal_Partners.xls, Pre_Synaptic_Neuronal_Partners.xls, and Gap_Junction_Partners.xls. This data set describes all presynaptic neurons and their targets, all postsynaptic neurons and their inputs and all gap junctions. This website was referred to by the

paper from Chen et al. (2006). Gap junctions were not included in this analysis for simplicity and maintenance of directional information, since gap junctions are bidirectional. The information in these three files was put into one file in a new format that could be processed by the graph computer programs used in this study. Synapses were provided in a file which first lists the 280 neurons in this analysis, followed by the number of synapses, followed one synapse per line.

Genetic Information

Genetic information corresponding to each of the neurons was taken from WormBase. In the cell browser, one may obtain an expression list for each neuron (http://www.wormbase.org/db/cell/cell.cgi?name=ADAL;class=Cell). This expression list does not correspond to all genes expressed in a neuron, but to published gene expression data which is in general derived from transcriptional or translational reporter constructs, or immunohistochemistry. These gene expression data are generally not reported in a time-specific manner, hence the analysis of gene expression data in this model must be done with the caveat that expression data is associated with a neuron born at a developmental stage at which the gene may not yet be expressed in vivo.

Combined File

The development stage program, as well as several of the pathway analysis programs here developed, uses a file which combines information about neurons, neuron categories, lineage strings, genes, and synapses. The format of this file is as follows: the number of neurons

is indicated, followed by a line which describes each neuron, its neuronal category, its lineage string, and its genes. The entire neuron list is then followed by the number of synapses and a list of the chemical synapses of the adult hermaphrodite.

Neuron Categories

The development stage program makes use of neuron categories. Each neuron is categorized into a main category of sensory, motor, or interneuron, and then may have up to two sub-categories. The categories were derived from the description field of the lineage file. Example sub-categories include 'chemo-sensory neurons,' 'amphid neurons,' 'ventral motor chord neurons.' If a neuron has only one sub-categorization, the second sub-categorization is marked by an 'x.' In the example above, ADAL is categorized as an interneuron, with the subcategory R, indicating it is a 'ring interneuron.'

Table 2. Neuron Categories.

Category	Abbreviation	Indication
Main	SN	Sensory
Main	MN	Motor
Main	IN	Interneuron
Sub	R	Ring
Sub	D	Dopaminergic
Sub	C	Cephalic
Sub	CH	Chemical, e.g chemosensory
Sub	A	Amphid
Sub	T	Touch
Sub	EC	Excretory Canal
Sub	P	Pharyngeal
Sub	H	Head
Sub	L	Labial
Sub	PH	Phasmid
Sub	DR	Dorsal
Sub	DC	Dorsal Cord
Sub	AD	Anterior Deirid
Sub	V	Vulval
Sub	PL	PLM
Sub	AL	ALM

III. Program

The developmental stages program was written in C++ using the GNU compiler (g++ version 4.1.1) and C++ Standard Template Library under Red Hat Linux 4.1.1-51. The execution platform was a 64 bit computer running GNU Linux 2.6.9 (System V) with dynamically linked shared libraries.

A. Data Structures

Since the development stages program reads the combined file, it must read information about neurons, neuron categories, lineage strings, and genes into a single data structure. The data structure used for this purpose is the neuron object. Since there are 280 neurons, the program uses an array of neuron objects, in this analysis this array is 280 cells long, but this number will automatically vary with the analysis being performed, and the length of the array is stored as a variable. The neuron object includes a data member, name, which contains the name of the neuron, which is the main field upon which the array of neurons is indexed. The object then contains data members for main category and two subcategories. The neuron object includes a field for lineage string. And then, since a set of genes are associated with each neuron, as its expression list, the final data member is a set of genes where each gene is its name represented as a string.

Vector of synapse objects

The combined file includes a list of synapses. This list is typically several thousand items

long, and will vary with the type of analysis being performed. Hence the data structure used to house the synapses is a vector of synapse objects. A vector is a resizeable array. Each synapse object has only three data members: neuron1, which is the 'from-neuron' or the presynaptic neuron, neuron2 which is the 'to-neuron' or the post-synaptic neuron, and an integer weight variable, which allows a weight or strength to be associated with a synapse. In this analyses this weight is always set to 1, as one of the programs that used repeatedly, shortestpath, requires the weight of the synapse to be set to 1.

Vector of neurons

As the program proceeds, a vector or resizeable array of neuron names (a neuron name is simply represented as a string) is constructed for each pass, i.e., each stage of development. Since the number of neurons varies, or more specifically grows, at each development stage, and because there are a varying number of neurons born with each new cell division, the neuron names are kept in a resizeable array, or a vector of neurons.

Vector of new neurons

Also kept for each development stage pass is a vector of the new neurons that are born at that development stage, or that number of cell divisions. This vector is a proper subset of the vector of neurons discussed above, and includes neurons with lineage strings equal to the lineage string length of the current development stage.

Categorical vectors

Vectors which categorize neurons per stage and new neurons per stage as either sensory, motor, or interneurons are also kept, for a more specific analysis. At this point in the project, only main category vectors are kept, but there is nothing preventing a future implemention of this public source code from creating more specific categorical vectors based also upon sub-categories, such as a chemosensory vector or mechanosensory vector.

B. Algorithm

The main pass or outer loop of the program involves going through the neuron object array and locating, for the first pass, all of the neurons with a lineage string of minimum length and constructing a neuron vector containing the names of those neurons. This vector contains the names of all neurons that are born at 11 cell divisions, that is, all neurons of the first development stage. For subsequent passes, two vectors are created. One is the vector of all neurons with lineage strings less than or equal to the current lineage string length (the minimum lineage string length is incremented by one during each pass), which represents all neurons born at this stage or already born, and one is the new neuron list of neurons born at specifically at this development stage.

During each pass, several categorical vectors are created as well. These are the sensory, motor, and interneuron vectors for all neurons already born or newly born on the one hand, and the sensory, motor, and interneuron vectors for only newly born neurons on the other hand.

At the end of each pass, the current pass vectors are destroyed, and recreated with a different number and set of neurons in the next phase. Again, vectors are used because they can be easily created and destroyed, sized and resized dynamically.

During each pass, a union set of all of the gene sets of all neurons per stage is created, as is a union set of all gene sets of only newly born neurons per stage. The former corresponds to all active genes, and the latter corresponds to only newly activated genes at this development stage. Further union sets are constructed for total sensory, motor, and interneurons per stage, and new sensory, motor, and interneurons per stage. These sets are made sensory, motor, or interneuron specific by taking the difference set from their counterparts. In addition, an intersection set of the sensory, motor, and interneurons is generated per development stage. This additional information allows identification of genes that are expressed in sensory neuron development or in motor neuron development.

Significantly, a totally novel piece of information is generated by this algorithm: a description of the development stage specific synaptic network. The development stage specific synaptic network is generated by picking out of the the total synapse list (of the adult animal) just those synapses where both the pre-synaptic and post-synaptic neuron belongs to the total neuron list for the current development stage. In this way the synaptic network grows as new neurons are born. Notably, the synapse list is constructed in a way where it can be read into the synaptic pathway analysis programs for development stage specific analysis, that is, it is a proper graph input file.

In addition to generating the entire synaptic network for each development stage, newly formed synapses are categorized by the algorithm into pre-synaptic (synapses where the new neuron synapses on a pre-existing, formerly born, neuron, that is, the new neuron is the pre-synaptic partner), into post-synaptic (synapses where a pre-existing, formerly born neuron synapses on a newly born neuron, that is, the new neuron is the post-synaptic partner), or into

'new connections' (where a newly born neuron synapses onto a newly born neuron). The logical complement, old neurons (where a previously existing neuron synapses on a previously existing neuron) is not computed as it may be obtained by simply looking at the total synapses of the previous development stage.

C. Results

It is the output of the development stage program which biologists are interested in analyzing. For each development stage, a list of newly born neurons is given, with that list being subcategorized into sensory, motor, or interneurons. A list of all neurons born up to that point of development is given as well, and this is also categorized into sensory, motor, or interneurons. For both the newly born neurons and all neurons born up to that point, the non-redundant set of genes corresponding to those neurons is given for developmental genetic analysis. Finally, a representation of the putative synaptic network at each stage of development is given, and for newly formed synapses, these synapses are subcategorized into pre-synaptic, post-synaptic, and new connections, indicating the role of the new neuron.

III. Biological Interpretation of Developmental Stage Output

The program generates data about which neurons are born at each new cell division, starting with cell division 11, and proceeding through the 16^{th} cell division. In each development stage, defined by the number of cell divisions, data is generated on which genes are active. Neurons per stage are unique and are in part what defines the stage. The same gene, however, may appear at multiple stages as one gene may be expressed in several neurons, each of which

may appear at a different stage. It is important to remember that a program has been built which generates data about each development stage; the value or accuracy of that data is in part dependent on how good the input file is. The analysis has been performed with a reasonable input file which describes the neurons, development lineage, neuron categories, genes per neuron, and synapses of the *C. elegans* nervous system. In the future, a run with an expanded gene list, for example, may yield additional insights. The development stages occur as described in the table below.

Table 3. Six stage developmental model.

Stage	Number of Cell Divisions	Number of Neurons	Number of Chemical Synapses
1	11	144	788
2	12	202 (58 new)	1537 (749 new)
3	13	212 (10 new)	1598 (61 new)
4	14	249 (37 new)	1924 (326 new)
5	15	276 (27 new)	2130 (206 new)
6	16	280 (4 new)	2199 (69 new)

The explosion of neuron births at 11 cell divisions appears to occur around 300 minutes (5 hours) into development in the embryonic stage. Most of the neural development appears to occur between 300 minutes (5 hours) and 800 minutes (13 hours and 33 minutes) still in the embryonic stage.

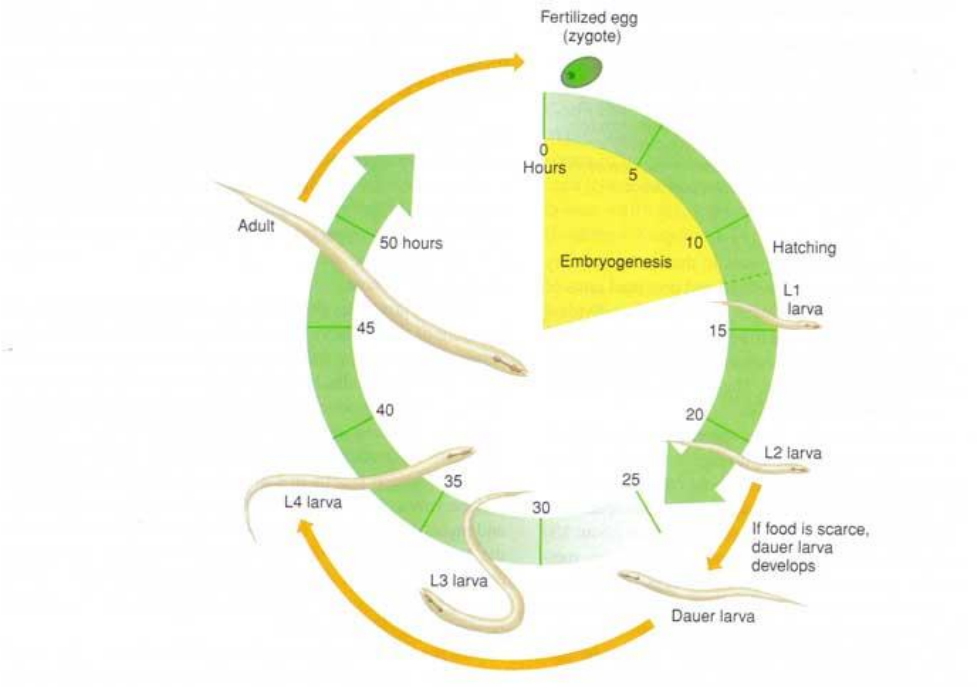

Figure N. ***C. elegans*** **developmental clock. Neural development occurs during embryogenesis.**

A. Initial Development Stage 1

This analysis indicates something notable about neuronal development. No neurons are born with fewer than 11 cell divisions. This indicates that neuronal development occurs in later stages of embryogenesis in *C. elegans*, that is, the zygote divides many times before the first neuron is born and many non-neuronal cells are born before neurons are born. Many neurons are born simultaneously at 11 cell divisions (some structural, muscle, and hypodermal cells are born before neurons). Specifically, in the first stage of neuronal development, 144 neurons, or 51% of the neurons of the adult, are born. If cell division were entirely symmetric at 11 cell divisions, 2048 cells could be born, so it isn't surprising that at 11 cell divisions a large number of neurons simultaneously appear. At stage one 788 synapses are formed. The program predicts synapses

by picking out of the adult hermaphrodite all synapses that only include neurons born at stage one. This represents about 35% of the synapses of the adult hermaphrodite, excluding electrical junctions. All neurons in this model organisms are descendents of the AB lineage; although, some neurons not in this model are descendents of the MS and C lineages.

The sudden burst in the number of neurons born at 11 cell divisions has been biologically verified by the embryonic lineage diagram at: http://www.wormatlas.org/userguidepics/general%20guides/lineage1.jpg. Here it can be seen that 144 neurons are born at 11 cell divisions, as descendents of the AB lineage. This includes 20 sensory neurons, 39 motor neurons, and 85 interneurons.

Table 4. Stage One Neurons

All Sensory neurons at this stage:
ADFL ADFR ADLL ADLR AFDL AFDR ALML ALMR ASGL ASGR ASHL ASHR AWAL AWAR AWBL AWBR AWCL AWCR PHAL PHAR
All Motor neurons at this stage:
DA1 DA2 DA3 DA4 DA5 DA6 DA7 DA8 DA9 DB1 DB2 DB3 DB4 DB5 DB6 DB7 DD1 DD2 DD3 DD4 DD5 DD6 PDA RIML RIMR RMDDL RMDDR RMDL RMDR RMDVL RMDVR RMED RMEL RMER RMEV URADL URADR URAVL URAVR
All Inter-neurons at this stage:
AIAL AIAR AIBL AIBR AIML AIMR AINL AINR AIYL AIYR ALA AVAL AVAR AVBL AVBR AVDL AVDR AVEL AVER AVG AVHL AVHR AVJL AVJR AVKL AVKR AVL BAGL BAGR BDUL BDUR DVA DVB IL2DL IL2DR IL2L IL2R IL2VL IL2VR PVPL PVPR PVQL PVQR PVT RIAL RIAR RIBL RIBR RID RIGL RIGR RIH RIPL RIPR RIR RIS RIVL RIVR RMGL RMGR SAADL SAADR SAAVL SAAVR SABD SIADL SIADR SIAVL SIAVR SIBDL SIBDR SIBVL SIBVR SMBVL SMBVR SMDDL SMDDR SMDVL SMDVR URBL URBR URYDL URYDR URYVL URYVR

At this stage the amphid chemosensory system is developing. ASGL/R, ASHL/R, AWAL/R, AWBL/R, and AWCL/R are amphid chemosensory neurons and AFDL/R is an amphid finger cell. These amphid chemosensory neurons are forming a circuit with the ventral cord and ring motor neurons. The DA series, DB series, and DD series motor neurons are all ventral cord motor neurons (the DD series is a ventral cord inhibitor), and the RM series and UR series are ring motor neurons. In keeping with this analysis, the interneurons born at this stage are ventral cord interneurons or ring interneurons. The AI, AV series are ventral cord interneurons. The PV series is a set of interneurons that extend from the ventral cord to the ring, and the SA, SI, SM, and UR series are ring interneurons. Hence, most of what is output at stage one from this program predicts the development of a circuit from amphid chemosensory neurons to ventral cord and ring motor neurons. The amphids are a prominent pair of chemosensory organs located on either side of the head. This seems to be the circuit that develops the earliest in the animal. In addition, there are two touch receptors that develop at this time: ALML and ALMR. And there are two phasmid chemosensory neurons that also develop: PHAL and PHAR. Also, some labial interneurons develop at this time: the IL series.

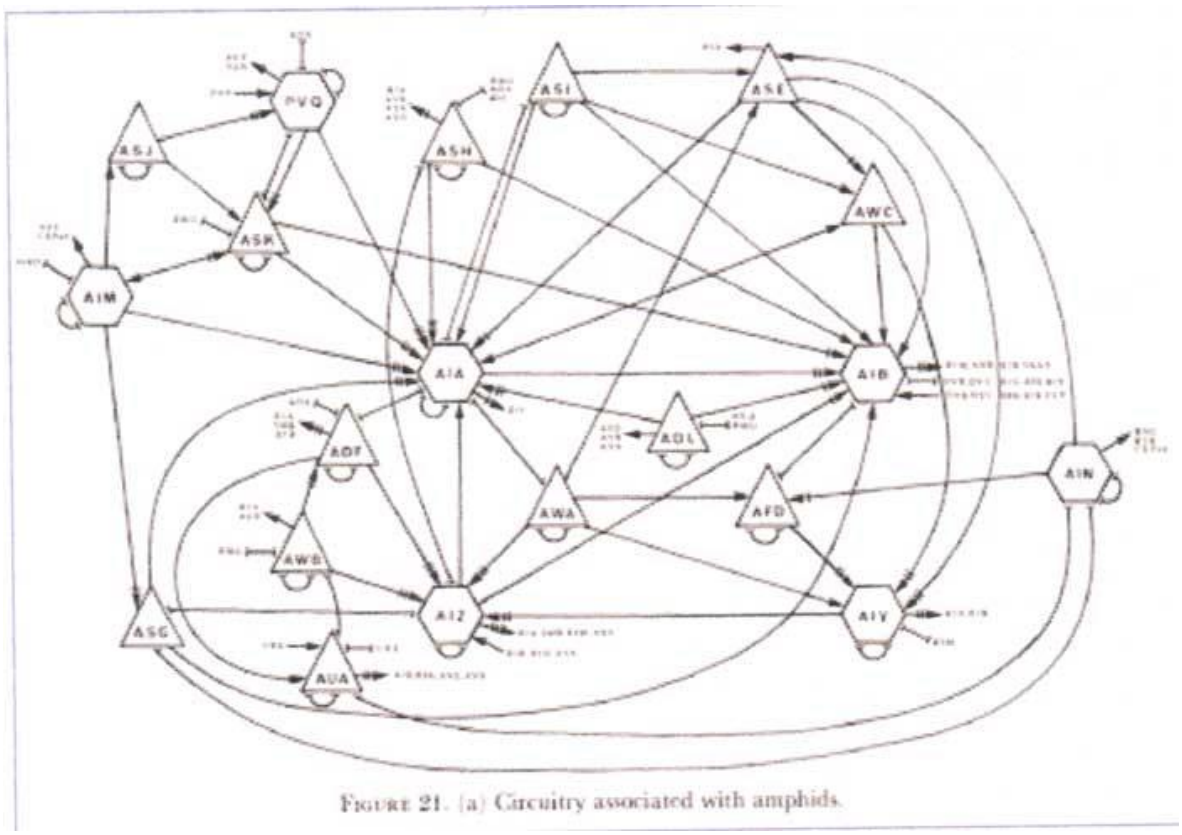

FIGURE 21. (a) Circuitry associated with amphids.

Figure 7. The amphid sensory circuit in the mature worm.

Sensory Development in Stage One

The developmental genetics of stage one: first the sensory specific genes, generated by the develstage program by taking the difference set of sensory genes from motor or interneuron genes at this stage.

The salient features of genes activated in sensory neurons at stage one are as follows. Some of these genes are involved in development of the olfactory sense, others in mechanosensation, others in the sensation of osmotic pressure, and some are involved in axon outgrowth and transport of these sensory cells. Genes involved in olfactory sense include: nsy-

1, the odr genes, the gpa genes, ceh-1 and ceh-37, srg-220, str1 and 2, and tbx-2. The odr genes encode guanylyl cyclases that localize to the flattened ciliated endings of the AWC neurons. The gpa genes encode the alpha subunit of a heterotrimeric GTP-ase involved in chemotaxis to water soluble odorants. The development of the amphid olfactory sense in stage one is activated by the genes: tbx-2, str-1, str-2, srg-220, odr-10, odr-7, and nsy-1. This sense is required for foraging and sexual activity. The sense of gustation is affected by gpc-1. Sensation of osmotic pressure is enabled by osm-9 and osm-10. Mechanosensation is enabled by several genes: egl-46, mec-10 and mec-4, mtd-1, mec-3, mps-1, unc-97. Genes affecting chemotaxis include gcy-12, ceh-36, srd-1, and bra-1. All of the genes identified by this program as being expressed in sensory neurons born at stage one, affect sensory processes. Clearly *C. elegans* is developing olfaction, gustation, some mechanosensation, sensitivity to osmotic pressure, and chemosensation at this stage.

Table 5. Sensory-specific genes of stage one:
bra-1 ceh-32 ceh-36 ceh-37 che-1 che-2 che-3 daf-11 daf-5 dyn-1 egl-43 egl-46 gcy-12 gpa-1 gpa-10 gpa-11 gpa-13 gpa-15 gpa-5 gpa-6 gpc-1 grk-2 lst-1 mec-10 mec-2 mec-3 mec-4 mec-6 mec-8 mig-2 mps-1 mtd-1 nhr-79 nid-1 nlp-10 nlp-14 nlp-7 nlp-8 nlp-9 nsy-1 ocr-1 ocr-2 odr-1 odr-10 odr-3 odr-7 osm-10 osm-9 pat-4 ptl-1 sgk-1 sra-13 srb-6 srd-1 sre-1 srg-13 srh-220 sro-1 srp-2 str-1 str-2 tag-97 tba-1 tbx-2 unc-32 unc-73 unc-97 ver-2

Table 6. Description of sensory genes active in stage one.

Gene	Description
Nsy-1	Nsy-1 encodes a MAP kinase kinase that leads to assymetric olfactory neuron fates affecting chemotaxis
Odr-10, Odr-7	Odr-10 and 7 encode a seven transmembrane odorant receptor used in AWAL/R, particularly for detecting a diacetyl odorant
Gpa-10	Gpa-10 is expressed in ADFL/R
Gpa-11	Gpa-11 is expressed in the ADLL/R and ASHL/R amphid sensory neurons.
Gpa-13, Gpa-15	Gpa-13 and gpa-15 are expressed in ADLL/R and ASHL/R but also in AWCL/R and PHAL/R.
Gpa-5, gpa-6	Gpa-5 and gpa-6 are expressed in AWAL/R.
Srg-220	Srg-220 encodes an olfactory G-protein coupled receptor.
str-1, str-2	str1 and str2 encode a seven transmembrane olfactory receptor
tbx-2	Tbx-2 encodes a T-box transcription factor required for adaptation to attractive odorants sensed by the AWCL/R amphid neurons.
gpc-1	gpc-1 encodes a heterotrimeric guanine nucleotide binding protein gamma subunit expressed in sensory neurons involved in taste adaptation.
Osm-9, osm-10	osm-9 and osm-10 are active in ASHL/R and PHAL/R, by osmotic signaling from the calcium ion channel which these genes encode.
egl-46	egl-46 encodes a zinc-finger protein that affects process formation in touch cells, particularly ALML/R which appear at this stage
Mec-10, mec-4	Mec-10 and mec-4 encode a amiloride-sensitive sodium channel protein (degenerin) required to sense gentle mechanical stimuli, present in ALML/R
Mtd-1	Mtd-1 affects the touch sensitivity of mec-6, and mec-6 physically interacts with mec-4 for punctate expression along the touch receptor neural process
Mec-3	Mec-3 encodes a LIM homeodomain transcription factor required for the maturation of mechanosensory neurons.
Mps-1	Mps-1 is required for body touch sensation; it is a MinK-related peptide that forms ion channels in ADFL/R.
Unc-97	Unc-97 is important for the function of mechanosensory neurons
gcy-12	gcy-12 encodes a guanylate cyclase that affects chemotaxis in PHAL/R phasmid neurons
ceh-36	ceh-36 affects chemosensation in AWCL/R neurons
Srd-1	srd-1 is a seven transmembrane chemosensory receptor
Bra-1	bra-1 regulates daf genes such as daf-5 and daf-11 which suppress dauer formation and regulate chemosensation in AWCL/R neurons

It was noted that the main motor development at stage one was of ventral cord and ring motor neurons. Both the difference set and all neuron-specific genes found in motor neurons at this stage were used for this analysis. Genes underlying this development, that were identified by the program as activated at this stage, include genes that affect specific motor neurons, genes that affect forward and backward locomotion, motor neuron genesis and axonogenesis. First genes that affect specific motor neurons include Chl-6, Flp-21, Kus-1, Flp-18 , Vab-7, Unc-4, Unc-47 (see table below for explanation). Secondly, locomotion is being developed at this stage and hence the program finds a number of genes activated which affect motor neuron locomotion. Genes affecting normal forward locomotion include unc-5, a netrin receptor and goa-1, and kus-1. Change in locomotor direction in response to a sensory cue is affected by glr-1 and 2, osmotic avoidance by kus-1, and foraging behavior by nmr-1. Unc-8 is responsible for the sinusoidal locomotion characteristic of *C. elegans*. Whereas, unc-53 and ace-3 affect backward locomotion. Finally, some genes active at this stage affect motor neurogenesis and axonogenesis. These include unc-40 and unc-5 which guide dorsal ventral cell and axon migrations, and unc-129 which encodes a TGF molecule required for proper axon guidance of commissural motor neurons. Mig-13 is expressed in ventral cord motor neurons and encodes a single pass transmembrane protein that affects the positioning of neuroblasts and their descendents. Genes identified as active at this development stage by the develstage program

Table 7. Motor-specific genes
ace-3 clh-6 daf-9 itr-1 mig-13 rig-5 smp-2 unc-129 unc-4 vab-7
Motor Neuron Genes at this stage
ace-2 ace-3 ahr-1 avr-15 cam-1 cav-1 cdc-42 ced-10 ceh-10 ceh-17 ceh-6 cfi-1 cha-1 chd-3 clh-6 csk-1 daf-9 dbl-1 dop-1 dop-2 egl-4 egl-5 fbxb-103 flp-18 flp-21 glr-1 glr-2 glr-4 glr-5 goa-1 ina-1 itr-1 kvs-1 lim-4 lim-6 lin-35 lin-53 mgl-1 mgl-2 mig-13 nmr-1 nmr-2 npr-1 odr-2 ptp-3 qui-1 rig-5 sax-3 ser-2 set-2 slt-1 smp-2 src-1 tax-6 unc-129 unc-17 unc-25 unc-4 unc-40 unc-42 unc-47 unc-5 unc-53 unc-8 unc-86 vab-7

participate in the development of the ventral motor neuron network which responds to chemosensory neuron input.

Table 8. Description of motor neuron genes in stage one.

Gene	Description
Chl-6	Chl-6 which affects GABA packaging in RMEL/R.
Flp-21	Flp-21 is an amide-related neuropeptide that affects URAL/R motor neurons and regulates social as opposed to solitary feeding behavior.
Kus-1	Kus-1 is an A-type potassium channel found in PDA motor neurons, which affects normal locomotion and osmotic avoidance (remember that genes for the sense of osmotic pressure have been activated at this stage).
Flp-18	Flp-18 encodes an FMRM amide-related neuropeptide active in RIML/R motor neurons and also in the AVAL/R, AIYL/R, and RIGL/R interneurons developed at this stage.
Vab-7	Vab-7 encodes a homeodomain protein required for DB motor neuron identity and posterior DB axonal polarity.
Unc-4	Unc-4 also encodes a homeodomain protein required for the identity of DA and VA motorneurons.
Unc-47	Unc-47 encodes a transmembrane GABA transporter expressed in D type motor neurons (transciptionally regulated by the unc-30 homeodomain protein)
unc-5	unc-5, a netrin receptor
nmr-1	nmr-1, an NMDA-type ionotropic glutamate receptor
Unc-8	Unc-8 is an amiloride sensitive cation selective channel responsible for the sinusoidal locomotion
unc-53, ace-3	unc-53 and ace-3, which encodes an acetylcholinesterase, affecting backward locomotion
unc-40, unc-5	unc-40 and unc-5 encode a netrin receptor required to guide dorsal ventral cell and axon migrations
unc-129	unc-129 encodes a TGF-beta secreted growth factor signaling molecule required for proper axon guidance of commissural motor neurons
Mig-13	Mig-13 is expressed in ventral cord motor neurons and encodes a single pass transmembrane protein that affects the positioning of neuroblasts and their descendents

Interneuron Development in Stage One

The purpose of interneurons developed at this stage is to connect the amphid and other mostly chemosensory neurons of this stage to the ventral motor neurons of this stage. The genes associated with these interneurons serve diverse purposes. Fax-1 affects fasciculation of axons of the interneurons AVBL/R, AVAL/R, AVEL/R, and AVKL/R. Flp-1 affects sinusoidal movements in the interneurons AVKL/R, AVAL/R, AVEL/R, RIGL/R, RMGL/R, AIYL/R, and AIAL/R. Some of these genes encode ion channels: ggr-1, ggr-2, glr-3 and 6 encode an ion channel that affects the RIAL/R interneurons. Gpa-16 affects spindle position and orientation in the embryo, in developing interneurons BDUL/R and PVC in the next stage. Hlh-14 encodes a transcription factor required for the development of the PVC interneuron, whereas kal-1 affects the development of AIYL/R and RIDL/R interneurons. Lad-2 regulates axon guidance and morphogenesis in the SMDL/R and SAAL/R interneurons. Ser-4 encodes a receptor expressed in RIBL/R, RISL/R, PVT, and DVA interneurons. Snt-1 encodes a synaptic vescicle protein involved in the DVB interneuron. And sra-11 encodes a receptor implicated in olfactory imprinting in the AIAL/R, AIYL/R and AVBL/R interneurons.

Table 9. Inter-neuron specific genes of stage one:
ceh-24 ceh-43 cnb-1 daf-14 deg-1 eat-20 egl-36 fax-1 flp-1 gcy-33 gcy-35 ggr-1 ggr-2 glr-3 glr-6 gly-18 gpa-16 gpa-9 hen-1 hid-1 hlh-14 igcm-2 kal-1 lad-2 let-60 nhr-22 nlp-11 nlp-6 oig-1 oig-3 opt-3 rig-3 ser-4 smp-1 sng-1 snt-1 sra-10 sra-11 sup-9 tba-2 trp-1 unc-6 unc-71 unc-93 vab-8 ver-3 wrk-1 zig-1 zig-2 zig-3 zig-4 zig-5 zig-8

Table 10. Description of interneuron genes of stage one.

Genes	Description
Fax-1	Fax-1 affects fasciculation of axons of the interneurons AVBL/R, AVAL/R, AVEL/R, and AVKL/R
Flp-1	Flp-1 encodes small neuromodulatory peptides that affect sinusoidal movements in the interneurons AVKL/R, AVAL/R, AVEL/R, RIGL/R, RMGL/R, AIYL/R, and AIAL/R.
Ggr-1, ggr-2	ggr-1 and ggr-2 encode a GABA/Glycine ligand gated chloride channel expressed in the ventral cord interneurons AIBL/R, PVR, PVQL/R, AVHL/R, and SMDVL/R
Glr-3, glr-6	glr-3 and 6 encode an AMPA ionotropic glutamate ligand gated ion channel that affects the RIAL/R interneurons
Gpa-16	Gpa-16 encodes a heterotrimeric GTPase that affects spindle position and orientation in developing interneurons BDUL/R and PVL in the next stage
Hlh-14	Hlh-14 encodes a helix-loop-helix transcription factor required for the development of the PVC interneuron
Kal-1	Kal-1 encodes a cell surface protein that affects the development of AIYL/R and RIDL/R interneurons
Lad-2	Lad-2 encodes an immunoglobulin adhesion molecule that regulates axon guidance and morphogenesis in the SMDL/R and SAAL/R interneurons
Ser-4	Ser-4 encodes a metabotropic serotonin receptor expressed in RIBL/R, RISL/R, PVT, and DVA interneurons and the retrovesicular ganglion
Snt-1	Snt-1 encodes synaptagmin a synaptic vescicle protein involved in neurotransmitter release and reuptake in the DVB intern
Sra-11	Sra-11 encodes a transmembrane receptor implicated in olfactory imprinting in the AIAL/R, AIYL/R and AVBL/R interneurons

Genes common to sensory neurons, motor neurons, and interneurons at this stage was generated by taking the intersection set of sensory, motor, and interneuron genes at this development stage. These genes are expected to have more general, for example, regulatory function than the sensory, motor, or interneuron specific genes. These genes are involved in neuronal migrations, axonogenesis, differentiation and cell fate determination, and encoding neurotransmitters. Neuronal migrations are enable by genes such as ina-1, , cam-1, and unc-40, required to guide dorsal-ventral neurogenesis. Several genes affect neuron fate: lim-4 is required for the differentiation

Table 11. Genes common to sensory neurons, motor neurons, and interneurons at stage one:
ace-2 cam-1 cav-1 cdc-42 ced-10 chd-3 csk-1 dop-1 egl-4 fbxb-103 goa-1 ina-1 lim-4 lim-6 lin-35 lin-53 npr-1 odr-2 ptp-3 set-2 src-1 tax-6 unc-40 unc-42 unc-8 unc-86

of AWBL/R chemosensory neurons, RMDL/R and RMEV motor neurons, and SAA and SIA interneurons; unc-42 specifies the fate of ASH sensory neurons, AVA, AVD, and AVE interneurons, and some motor neurons; and unc-86 is responsible for the determination of diverse cell lineages. Some common genes affect axonogenesis: unc-40 affects axon migrations, odr-2 enriches the axons of sensory, motor, and interneurons. Other genes specify neurotransmitters such as dop-1 and ace-2.

Table 12. Description of genes common to stage one.

Gene	Description
ina-1	ina-1 encodes an integrin subunit
cam-1, unc-40	cam-1 and unc-40 encode a netrin receptor require to guide dorsal-ventral neurogenesis
Lim-4	lim-4 encodes a homeodomain protein required for the differentiation of AWBL/R chemosensory neurons, RMDL/R and RMEV motor neurons, and SAA and SIA interneurons.
Unc-42	unc-42 is a homeodomain protein that specifies the fate of ASH sensory neurons, AVA, AVD, and AVE interneurons, and some motor neurons.
Unc-86	unc-86 encodes a POU-type homeodomain protein responsible for the determination of diverse cell lineages
Unc-40	netrin receptor
Odr-2	odr-2 is a glycosylated phosphatidylinositol linked signal protein which enriches the axons of sensory, motor, and interneurons
Dop-1	dop-1 is a dopamine receptor expressed in mechanosensory neurons, motor neurons, and the RIS interneurons
Ace-2	ace-2 encodes an acetylcholinesterase involved in the termination of cholinergic nerve transmission

Synapse Development in Stage One

There are 788 unique chemical synapses that form out of the 144 neurons that are born at 11 cell divisions in stage one. For the specific and complete list of synapses see the data file "develstage1.txt" on the CD.

B. Development Stage 2

Sensory Development in Stage Two

The anterior deirid sensory neurons ADEL and ADER are developed. Continued development of the amphid chemosensory neurons now projecting into the ring from the ventral ganglion with ASEL and ASER and ASIL, ASIR, ASJL, ASJR, and ASKL and ASKR. PHBL and PHBR are chemosensory phasmid neurons, furthering the development of PHAL and PHAR.

Table 13. Sensory neurons at stage two.
ADEL ADER ASEL ASER ASIL ASIR ASJL ASJR ASKL ASKR PHBL PHBR

By comparing the newly activated gene list with the sensory specific gene list, the newly activated sensory genes at stage two can be determined.

The daf genes, daf-28 inhibits dauer formation, expressed in ASJ and ASI neurons, born at this stage. Ida-1 is required for daf-28 to function. Daf-7 affects dauer formation in ASI sensory neurons, but also affects egg-laying. Srg-8 is also a chemosensory receptor in ASK neurons that affect the sensory regulation of egg-laying, as well as chemotaxis to lysine. Several genes are activated at this stage which are involved in oxidative stress response: gcs-1 is involved with stress to heat or herbicide; skn-1 functions in ASI neurons in the oxidative stress response pathway. The gcy genes function in chemosensory signal transduction. Lsy-6 controls the assymetrical expression of the gcy receptors in ASEL/R. Gpa-4 is involved in chemotaxis to water soluble odorants, it interacts with gpb-1 and 2, which are also predicted as activated at this stage, and affect egg-laying. The nlp series which encode a neuropeptide expressed in PHBL/R, which may have a non-neural function. Unc-3 and 37 encode transcription factors.

Table 14. Newly activated sensory genes at stage two.
daf-28 daf-7 gcs-1 gcy-5 gcy-6 gcy-7 gpa-4 lsy-6 nlp-24 nlp-27 nlp-5 skn-1 sra-7 sra-9 srg-8 str-3 unc-3 unc-37

Table 15. Descriptions of newly activated genes at stage two.

Gene	Description
Daf-28	daf-28 encodes a beta-type insulin that inhibits dauer formation. It is expressed in ASJ and ASI neurons
Ida-1	Ida-1, a tyrosine phosphatase, identified by this program as a common gene in this phase, is required for daf-28 to function
Daf-7	Daf-7 encodes a growth factor that affects dauer formation in ASI sensory neurons, but also affects egg-laying
Srg-8	Srg-8 is a chemosensory receptor in ASK neurons that affect the sensory regulation of egg-laying, as well as chemotaxis to lysine
gcs-1	gcs-1 is a gamma glutamine cysteine synthetase, expressed in ASI sensory neurons,
Skn-1	skn-1 encodes a transcription factor that functions in ASI neurons in the oxidative stress response pathway.
Gcy genes	gcy genes encode a natriuretic guanylate cyclase peptide receptor expressed in ASEL/R
Lsy-6	Lsy-6 controls the assymetrical expression of the gcy receptors in ASEL/R; it encodes a microRNA involved in the functional lateralization of the ASE chemosensory neurons
Gpa-4	Gpa-4 encodes a heterotrimeric GTP-ase, expressed in ASI
Nlp series	nlp series which encode a neuropeptide expressed in PHBL/R
Str-3	Str-3 encodes a seven transmembrane chemosensory receptor in ASI.

Motor Development in Stage Two

Notable in development stage two is the appearance of HSNL/R which has been studied extensively by Kong Shen. HSN motor neurons innervate vulval muscles and stimulate egg-laying, but first migrate along the ventral cord neurons that were developed in stage one. It should be noted that at stage two OLLL/R, OLQDL/R and OLQVL/R outer labial motor neurons develop at this stage for egg-laying behavior. This information in concert with the fact that the inner labial neurons IL2DL/R, IL2L/R, and IL2VL/R were developed in stage one, sets the stage for the vulval egg-laying circuit in stage two, with the understanding that the complete formation and innervation of the vulva doesn't occur until adulthood.

Table 16. New motor neurons at stage two:
HSNL HSNR OLLL OLLR OLQDL OLQDR OLQVL OLQVR RMFL RMFR RMHL RMHR VA1 VD1

In addition, there is some further development of the ventral cord with VA1 and VD1, which adds fine motor control with a reciprocal inhibitor, and the addition of four ring motor neurons: RMFL/R and RMHL/R.

Consider the newly activated motor gene list. Several genes are involved with the migration of HSNL/R, critical in vulval development: mig-1 encodes a receptor required for the proper migration of HSNL/R, as does ham-2, while syg-1 is required for HSNL/R to specify synaptic specificity with vulval muscles and the VC motor neurons. Ham-2 interacts with egl-5 and unc-86, genes both formerly expressed in stage one, according to this program. Gar-2 is expressed in HSN and ventral cord neurons. Other genes are involved with vulval development and egg-laying behavior. The is expressed in the labial sensory neurons, as well as the amphid sensory neurons and nerve ring. Inx-4 and nhx-5 are

Table 17. Newly activated motor genes at stage two:
bbs-5 cdh-3 che-13 clh-3 eat-16 flt-1 gar-2 gpb-2 grd-6 gsa-1 ham-2 hbl-1 ins-11 ins-2 ins-22 ins-23 ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 inx-4 mab-23 mig-1 nhx-5 ocr-4 osm-5 rgs-2 sem-4 syg-1 unc-14 unc-51 unc-76

expressed in neurons of the developing vulva. Genes implicated in egg-laying behavior include hbl-1, ngs-2, a gene that regulates egg-laying behavior of re-fed animals after starvation, eat-16 and gpb-2 , which are proteins responsible for pharyngeal pumping as well as egg-laying; gsa-1 is involved in movement and egg-laying, and sem-4 which affects sex myoblast migration and egg-laying behavior. Other genes here listed are involved in neurotransmission and axonogenesis.

Table 18. Descriptions of newly activated motor genes at stage two.

Gene	Description
Ham-2	ham-2 is a zinc-finger protein
Gar-2	Gar-2 encodes a seven transmembrane acetylcholine receptor expressed in HSN and ventral cord neurons
Ins series	ins series encodes an insulin-like peptide expressed in the labial sensory neurons
Inx-4	Inx-4 encodes an innexin channel expressed in neurons of the developing vulva
Nhx-5	nhx-5 encodes a sodium/proton exchanger expressed in vulval neurons
Hbl-1	hbl-1 encodes a zinc-finger protein
Ngs-2	ngs-2, a GTPase activating protein that regulates egg-laying behavior of re-fed animals after starvation
Gsa-1	gsa-1 is a heterotrimeric G protein subunit involved in movement and egg-laying

Interneuron Development in Stage Two

The interneurons in this stage need to support the major development of egg-laying circuitry. IL1DL/R, IL1L/R, and IL1VL/R are all inner labial interneurons. Head circuitry

Table 19. New inter-neurons at stage two:
ADAL ADAR AIZL AIZR AUAL AUAR CEPDL CEPDR CEPVL CEPVR FLPL FLPR IL1DL IL1DR IL1L IL1R IL1VL IL1VR LUAL LUAR PVCL PVCR RICL RICR RIFL RIFR SABVL SABVR SMBDL SMBDR URXL URXR

in the ring continues to develop with ADAL/R, RMFL/R, RMHL/R, RICL/R, RIFL/R, SABVL/R, SMBDL/R, and URXL/R. Amphid and cephalic development continues with AIZL/R and AUAL/R, and the dopaminergic CEPDL/R and CEPVL/R. PVCL/R develops in a ventral cord lumbar ganglion innervating the VB and DB motor neurons.

Newly activated interneuron genes at stage two include des-2, which regulates fast action acetylcholine at neuromuscular junctions. Several genes here affect the URXL/R ring interneurons: the gcy series are expressed in URXL/R and also in PQR, AQR which are born at that next stage, and are implicated in fluid homeostasis; rig-1 and rig-6 are expressed in the ring interneurons, and required for reproduction and positive regulation of body size; gpa-8 is expressed in URX interneurons as well. Flr-4 affects sensitivity to fluoride ions, and is expressed in AUAL/R.

Table 20. Newly activated interneuron genes at stage two:
des-2 flr-4 gcy-32 gcy-34 gcy-36 gcy-37 gpa-8 mab-21 mab-9 mdl-1 rig-1 rig-6 sre-37

Table 21. Description of newly activated genes in interneurons of stage two.

Gene	Description
Des-2	des-2 is a nicotinic acetylcholine receptor that regulates fast action acetylcholine at neuromuscular junctions.
Gcy series	gcy series encode a guanylate cyclase
Rig-1, rig-6	rig-1 and rig-6 encode a protein similar to the vertebrate contactin
Gpa-8	gpa-8 is a GTPase expressed in URX interneurons
Flr-4	Flr-4 encodes a serine/threonine kinase that affect sensitivity to fluoride ions

Genes Common to Sensory, Motor, and Interneurons at Stage Two

Genes common to neurons at stage two include genes which affect axonogenesis, development of the vulva and egg-laying, development of the pharynx, and various sensory functions. Cam-1, kal-1, and unc-40 affect axonogenesis: cam-1 affects locomotion, cell migration, and axonogenesis; kal-1 is required for epithelial morphogenesis and axon branching in AIY, AIZ, RID, ASI, PVM, DVB, DVC, and PDB; whereas unc-40 affects dorsal ventral neuron and axon migrations and QL and QR neuroblast polarization. Development of the vulva is affected by chd-3 affects notch dependent vulval development, and lin-35 which affects vulva and pharynx development; ida-1 is required for the neuropeptidergic control of egg-laying in the vulva, nerve ring, and ventral cord, and ina-1 affects axon fasciculation of the head, pharynx and egg-laying apparatus. Some common genes affect sensory functions: npr-1 regulates anxiety, food consumption, and pain sensation, as well as social versus solitary feeding behavior, and osm-9 is a calcium channel which moderates response to osmotic pressure and mechanical stimuli.

Table 22. Genes common to new neurons at stage two:
cam-1 cav-1 cdc-42 ced-10 chd-3 che-2 eat-4 ida-1 ina-1 kal-1 lin-35 lin-53 npr-1 osm-6 osm-9 set-2 unc-40

Table 23. Description of genes common to new neurons in stage two.

Gene	Description
Cam-1	cam-1 is a tyrosine kinase receptor affecting locomotion, cell migration, and axonogenesis
Kal-1	kal-1 is a cell surface protein with a fibronectin domain required for epithelial morphogenesis and axon branching in AIY, AIZ, RID, ASI, PVM, DVB, DVC, and PDB
Unc-40	unc-40 is a netrin receptor which affects dorsal ventral neuron and axon migrations and QL and QR neuroblast polarization
Chd-3	chd-3 is a Phd finger protein of the SNF2 family that affects notch dependent vulval development
Ida-1	ida-1is a tyrosine phosphatase receptor

Ina-1	ina-1 is an integrin which affects axon fasciculation of the head, pharynx and egg-laying apparatus
Osm-9	osm-9 is a calcium channel which moderates response to osmotic pressure and mechanical stimuli

Synapse Development in Stage Two

749 new chemical synapses develop at stage two (the specific synapses are available in the file "develstage2.txt" on the CD). This means there were 1,537 total synapses at stage two. 387 of these new connections are pre-synaptic with respect to the new neurons, i.e., neurons of stage one synapses on the new neurons of stage two; 236 are post-synaptic, i.e., the new neurons of stage two synapse on neurons born at stage one; and 126 represent new connections: new neurons of stage two synapsing on new neurons of stage two. Which specific synapses fall into each of these groups is presented in the output of the development stage program but the data is too voluminous to be presented in this paper. But, this information is available in the CD attached to this paper.

C. Development Stage 3

Sensory Development in Stage Three

Stage 3 represents neurons born at 13 cell divisions. These include the PLML and PLMR posterior lateral microtubule cell touch receptors. Hence, at this stage there is further

development of the mechanosensory system that started with ALML and ALMR in the first stage.

In keeping with the fact that two new mechanosensory neurons are added in stage three, several mec genes are active. Mec-3 regulates the differentiation and maturation of mechanosensory neurons, initiated by unc-86 (from stage one). Mec-2 and mec-4 are responsible for gentle touch sensation. Mec-6 interacts physically with mec-4 for punctuate expression. Several genes encode tubulins for structural integrity in mechanosensory neurons: these include tba-1, tba-2, and ptl-1. Some genes here expressed repress touch neuron fates in non-mechanosensory cells, such as egl-46 that inhibits touch cell fate in stage two FLP cells, and pag-3 which represses the expression of mec-3 and mec-7 in the stage one BDUL/R. A couple of genes are involved in thermosensation: pkc-1 affects thermotaxis and sensation of volatile and soluble compounds, and eat-4 which affects thermosensation.

Table 24. New sensory neurons and sensory neuron genes at stage three.
New sensory neurons at this stage:
PLML PLMR
New sensory-specific genes:
daf-1 deg-3 eat-4 egl-46 egl-5 glr-8 gpa-16 mec-2 mec-3 mec-4 mec-6 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1 ptl-1 rhgf-1 sax-7 tba-1 tba-2 unc-73 unc-97

Table 25. Description of sensory specific genes active in stage three (as well as stage one).

Gene	Description
Mec-3	Mec-3 is a LIM family homeodomain transcription factor which regulates the differentiation and maturation of mechanosensory neurons, initiated by unc-86
Mec-2	Mec-2 is a stomatin responsible for gentle touch sensation
Mec-4	mec-4 is an amiloride sensitive sodium channel responsible for gentle touch sensation
Ptl-1	ptl-1 is a microtubule binding protein which promotes microtubule assembly and stability
Egl-46	egl-46 is a zinc finger protein that inhibits touch cell fate in stage two FLP cells
Pag-3	pag-3 is a zinc-finger protein which represses the expression of mec-3 and mec-7 in the stage one BDUL/R
Pkc-1	pkc-1 is a serine-threonine kinase which affects thermotaxis and sensation of volatile and soluble compounds

Eat-4	eat-4 is a glutamate transporter that affects thermosensation.

There are notably no new motor neurons at this stage. Hence, these new touch receptors connect up with the existing network of ventral cord and ring inter-neurons.

Interneuron Development in Stage Three

A few new interneurons are developed at stage 3. These include ALNL and ALNR, interneurons that run along side the touch receptors ALML and ALMR.

Table 26. New inter-neurons born at stage three:
ALNL ALNR AQR PQR PVWL PVWR

AQR is a neuron that projects into the ring and is not part of a sensilium. PQR is the posterior counterpart to AQR, and projects into the pre-anal ganglion. There is further posterior ventral cord development with the interneurons PVWL and PVWR.

Interneuron specific genes in stage three implement miscellaneous functions including axonogenesis, locomotion, feeding behavior, and various sensory functions.

Table 27. Interneuron-specific genes active in stage three
ace-3 ceh-13 ceh-14 cha-1 che-2 che-3 cki-1 daf-28 eat-20 egl-2 egl-36 egl-4 gcy-32 gcy-34 gcy-36 gcy-37 gpa-10 gpa-8 kal-1 lad-2 npr-1 odr-2 osm-6 ser-2 tax-2 unc-103 unc-17 unc-53

Axonogenesis is accomplished by kal-1, lad-2, and odr-2. Locomotion is affected by unc-53 and ace-3 which enable backward locomotion, and cha-1, that affects normal growth and locomotion. The pharynx and feeding behavior is affected by eat-20, expressed in the pharynx, egl-4 which affects feeding behavior and egg-laying, and the gcy series, expressed in neurons connected to the pseudocoelom, which plays a role in fluid homeostasis and feeding behavior. Genes involved in sensation include che-2 and che-3,

responsible for the structural integrity of sensory cilia, ceh-14, required for the specification of the AFD thermosensory neurons, and npr-1, involved in pain sensation.

Table 28. Description of interneuron specific genes active in stage three.

Gene	Description
Kal-1	kal-1 is a cell surface protein that affects axon branching
Lad-2	lad-2 is an adhesion molecule that affects axon guidance in ALN, PLN
Odr-2	odr-2 is a glycosylated phosphatidylinositol signal protein that is enriched in axons
Ace-3	ace-3 (an acetylcholinesterase) which enables backward locomotion
Cha-1	cha-1 is a choline acetyltransferase that affects normal growth and locomotion
Che-2	che-2 and che-3 are dynein heavy chains responsible for the structural integrity of sensory cilia
Ceh-14	ceh-14 is a LIM homeodomain protein required for the specification of the AFD thermosensory neurons

Synaptic Network in Stage 3

At stage 3, 61 new chemical synapses are formed, yielding a total of 1,598 synapses at this stage. 43 of these new synapses are presynaptic, i.e., involve neurons new to stage 3 connecting to neurons of previous stages; 17 are postsynaptic, i.e., neurons of previous stages synapse on neurons new to this stage, and there is only one new connection, a connection between a novel neuron and another novel neuron, at this stage. With 1,598 total synapses, the chemical synaptic network is 72.6% developed at this point. Most of this synaptic development, 69.8% of it, occurred in the first two development stages, indicating that when many neurons are born at 11 and 12 cell divisions, much of the chemical synaptic development occurs at this time as well. The entire stage 3 synaptic network is stored in the file “develstage-3.txt” on the CD.

D. Development Stage 4

Sensory Development in Stage Four

Sensory development in stage 4, or at 14 cell divisions continues with the development of more mechanoreceptors: AVM the anterior ventral microtubule cell touch receptor, and PVM the posterior ventral microtubule cell touch receptor.

Most notable in the developmental genetic analysis are the mec genes here expressed which are active in mechanosensory receptors. Mec-10 and mec-4 are required for gentle touch sensation; as is mec-2. The mec-6 product physically interacts with mec-4 degenerin for touch, and mtd-1 encodes a transmembrane protein that enhances the touch sensitivity of mec-6. Several genes identified by the develstage program here encode tubulins which are required for mechanosensation at the cell wall: mec-7, tba-1, and ptl-1. Two proteins are specifically expressed in AVM and PVM: gar-1 encodes a receptor in PVM, and gpa-16 encodes a GTPase found in AVM. The story here is that the genes the develstage program identified as sensory specific genes do in fact encode various aspects of mechanosensation, in keeping with the discovery that AVM and PVM mechanoreceptors are born at this stage.

Table 29. Sensory genes active in stage four.
dop-1 eat-4 egl-46 flt-1 gar-1 goa-1 gpa-16 mab-21 mec-10 mec-2 mec-3 mec-4 mec-6 mec-7 mec-8 mtd-1 pat-4 ptl-1 sax-7 tba-1 tol-1 unc-40 unc-97

Table 30. Description of sensory genes active in stage four.

Gene	Description
Mec-10, Mec-4	Mec-10 and mec-4 encode an amiloride-sensitive sodium channel required for gentle touch sensation
Mec-2	mec-2 which encodes a stomatin homolog for gentle touch.
Mec-7	mec-7 encodes a beta-tubulin
Tba-1	tba-1 encodes a tubulin
Ptl-1	ptl-1 encodes a microtubule binding protein

Gar-1	gar-1 encodes a G-protein linked acetylchoine receptor in PVM
Gpa-16	gpa-16 encodes an alpha subunit of a heterotrimeric GTPase

Motor Development in Stage Four

There is significant motor development in stage 4. The AS motor neuron series appears, ventral cord motor neurons that innervate dorsal muscles. The VC series appears completing the circuit initiated in Stage 2, providing ventral cord motor neurons that innervate the vulval muscles. And the VD series also appears, reciprocal inhibitors of ventral cord motor neurons, adding fine ventral body control. In addition two more motor neurons appear: PDB, a motor neuron in the dorsal cord and preanal ganglion, and VB2, another ventral cord motor neuron. Hence at this stage dorsal motor control develops, with continuation of the circuitry for egg-laying.

Table 31. Motor neurons born in stage four.
AS1 AS2 AS3 AS4 AS5 AS6 AS7 AS8 AS9 AS10 AS11 PDB VB2 VC1 VC2 VC3 VC4 VC5 VC6 VD2 VD3 VD4 VD5 VD6 VD7 VD8 VD9 VD10 VD11 VD12 VD13

The motor-specific genes identified by this program have several functions. Ida-1, lin-29, and unc-42 affect the VC vulval motor neurons and egg-laying behavior. Unc-42 encodes a protein that regulates egg-laying as well as backward locomotion and mechanosensation. Other genes here encode neurotransmitters, such as unc-47, cat-1 which encodes a transporter of dopamine and serotonin which affects pharyngeal pumping of food and egg-laying, and unc-25.

Table 32. Motor-specific genes active in stage four.
cat-1 ced-10 dbl-1 del-1 exc-7 hmr-1 ida-1 lin-29 unc-25 unc-42 unc-47 vab-7

Table 33. Description of motor genes active in stage four.

Gene	Description
Ida-1	Ida-1 encodes a tyrosine phosphatase receptor found in VC motor neurons that affect egg-laying
Lin-29	Lin-29 encodes a zinc-finger transcription factor that affects the heterochronic development of the egg-laying system
Unc-42	unc-42 encodes a homeodomain protein that regulates egg-laying as well as backward locomotion and mechanosensation.
Unc-47	unc-47 encodes a GABA transporter
Cat-1	cat-1 which encodes a synaptic vescicle monoamine transporter of dopamine and serotonin which affects pharyngeal pumping of food and egg-laying
Unc-25	unc-25 that encodes a GABA neurotransmitter

Interneuron Development in Stage Four

A few novel interneurons appear at stage 4. AVFL/R are interneurons connecting the ventral cord and ring. The posterior ventral cord interneurons PVNL and PVNR appear. More ring projecting interneurons are SDQL and SDQR.

Table 34. Inter-neurons born in stage four.
AVFL AVFR PVNL PVNR SDQL SDQR

The interneuron genes encode neurotransmitters and are involved with axonogenesis. Cha-1 is required for locomotion, unc-17 is required in ventral cord cholinergic neurons. Lad-2 regulates axon guidance. Unc-73 affects D type motor neurons and amphid axon outgrowth.

Table 35. Inter-neuron-specific genes active in stage four.
ceh-14 cha-1 lad-2 ser-2 unc-17 unc-73

Table 36. Description of interneuron genes active in stage four.

Gene	Description
Cha-1	Cha-1 encodes an acetyltransferase that synthesizes acetylcholine, required for locomotion.
Unc-17	Unc-17 encodes a synaptic vescicle acetylcholine transporter required in ventral cord cholinergic neurons
Lad-2	Lad-2 encodes an immunoglobulin that acts as a neural adhesion molecule, regulating axon guidance

Unc-73	Unc-73 encodes a GNEF, guanine nucleotide exchange factor that affects D type motor neurons and amphid axon outgrowth.

Genes Common to Sensory, Motor, and Interneurons at Stage Four

Pag-3 is picked out by this program as the only gene occurring in both interneurons, mechanosensory neurons, and ventral cord motor neurons at this stage as it has multiple functions. Pag-3 encodes a zinc finger protein which prevents the repression of touch sensitivity in mechanoreceptors, while at the same time is responsible for normal locomotion in the ventral nerve cord.

The Synaptic Network in Stage 4

At development stage 4, there are 1,924 synapses in all, with 326 new synapses. 136 of these new synapses are presynaptic, i.e., these new neurons synapse on earlier stage neurons, 136 of these are postsynaptic, i.e., earlier stage neurons synapse on these new neurons, and 58 represent new connections, new neurons of this stage connecting to themselves. The entire stage 4 synaptic network is stored in the file "develstage4.txt" on the CD.

E. Development Stage 5

Sensory Development in Stage Five

In stage 5 dopaminergic neurons of the postderid sensillium are developed: PDEL and PDER. Striated rootlet neurons develop: PHCL and PHCR. As would be expected dop-2 is expressed at this stage: a dopamine receptor. cat-2 affects dopamine levels that in

Table 37. Sensory neurons born at stage five and their genes.
New sensory neurons at this stage:
PDEL PDER PHCL PHCR
New sensory-specific genes
cat-2 ceh-14 che-2 che-3 dat-1 dop-2 egl-36 gpa-16 ida-1 jkk-1 jnk-1 mab-23 osm-6 pkc-1 unc-40 unc-73

turn affect locomotory slowing in response to food. Dat-1 also encodes a plasma membrane dopamine transporter, expressed in PDEL/R. There seems to be some development of thermotactic behavior at this stage, with the expression of ceh-14, and pkc-1. Other genes affecting neurotransmissions include ida-1 that regulates pre-synaptic transmission, jkk-1 that affects synaptic vescicle localization, which is an activator of jnk-1, co-expressed at this stage. Mab-23 is a transcription factor that is required for the morphogenesis and patterning of posterior sensory neurons.

Table 38. Description of sensory specific genes at stage five.

Gene	Description
Cat-2	cat-2 encodes a putative tyrosine hydroxylase that affects dopamine levels that in turn affect locomotory slowing in response to food
Dat-1	Dat-1 also encodes a plasma membrane dopamine transporter, expressed in PDEL/R
Pkc-1	pkc-1 is a serine/threonine kinase affecting thermotaxis
Ida-1	ida-1 encodes a tyrosine phosphatase receptor that regulates pre-synaptic transmission
Jkk-1	jkk-1 is a MAP kinase kinase that affects synaptic vescicle localization

Motor Development in Stage Five

Probably the major development seen in stage 5 is the birth of VA and VB motor neurons, i.e. further development of the ventral cord motor neurons. These are both sets of

Table 39. Motor neurons born at stage five and their genes.
New motor neurons at this stage:
VA2 VA3 VA4 VA5 VA6 VA7 VA8 VA9 VA10 VA11 VA12 VB1 VB3 VB4 VB5 VB6 VB7 VB8 VB9 VB10 VB11
New motor-specific genes:
avr-15 dbl-1 del-1 exc-7 lin-29 mab-21 pag-3 unc-42 unc-6

motor neurons that innervate the ventral body muscles. Avr-15 mediates fast inhibitory glutamatergic neuromuscular transmission. Del-1 is involved with movement control. Mab-21 has multiple functions one of which is backward locomotion in the ventral cord. Pag-3 is expressed in ventral cord neurons required for neuroblast lineages in the ventral cord and for normal locomotion. And Unc-6 and unc-42 are required for axonal migration and axonal outgrowth in backward motor neurons, respectively. Ventral cord motor control is being extended and fine-tuned at stage 5.

Table 40. Description of motor neuron specific genes at stage five.

Gene	Description
Avr-15	Avr-15 encodes two glutamate-gated chloride channels that mediate fast inhibitory glutamatergic neuromuscular transmission.
Del-1	Del-1 encodes a degenerin sodium channel involved with movement control
Pag-3	Pag-3 encodes a zinc-finger protein expressed in ventral cord neurons required for neuroblast lineages in the ventral cord and for normal locomotion
Unc-6, unc-42	Unc-6 and unc-42 encode a netrin required for axonal migration and a homeodomain protein responsible for axonal outgrowth in backward motor neurons

Interneuron Development in Stage Five

PLNL and PLNR are interneurons that support the mechanosensory neurons developed in stage 3: PLML/R. The model in this study predicts that these interneurons

Table 41. Inter-neurons born at stage five and their genes.
New inter-neurons at this stage:
PLNL PLNR
New interneuron-specific genes:
cha-1 daf-1 egl-2 gcy-35 lad-2 odr-2 unc-17

develop after the sensory neurons they connect to. Cha-1 encodes a protein that synthesizes acetylcholine, critical to locomotion. Egl-2 is used in mechanosensation. Lad-2 regulates axon guidance in PLNL/R interneurons. Odr-2, responsible for odorant sense in AWC neurons is also expressed in other sensory neurons in which it plays a role in axon outgrowth.

Table 42. Description of interneuron specific genes at stage five.

Gene	Description
Cha-1	Cha-1 encodes a choline acetyltransferase that synthesizes acetylcholine, critical to locomotion
Egl-2	Egl-2 encodes a voltage-gated potassium channel which is used in mechanosensation
Lad-2	Lad-2 encodes an immunoglobulin cell adhesion molecule that regulates axon guidance in PLNL/R interneurons

Synapse Development in Stage Five

In stage 5, 206 new synapses appear, bringing the total synaptic network at this point to 2,130 synapses. 93 of these new synapses are pre-synaptic, 89 are post-synaptic, and 24 represent connections made between new neurons at this stage. The stage 5 synaptic network may be viewed in its entirety in the file “develstage5.txt” on the CD.

F. Development Stage 6

Interneuron Development in Stage Six

There is no further development of the sensory or motor systems in the final stage of

development. At this stage, four new interneurons appear. DVC is a late ring interneuron in the dorsalrectal ganglion. PVR is a ventral to ring interneuron. And PVDL/R are neuron lateral processes adjacent to the excretory canal. One development at this late stage seems to be innervation of the excretory canal. Several genes in the last phase precipitate this development: exc-7, for example, encodes a protein required for the formation of the excretory canal. Other functions include further development of innervation required for locomotion, feeding and egg-laying behavior, and neurogenesis and axonogenesis. Locomotion is affected by deg-3 and des-2 which are co-expressed; locomotion is also affected by goa-1 and jkk-1 (which activates jnk-1). Pkc-1 affects acetylcholine release at neuromuscular junctions, and the unc genes which encode netrins and ATPases, also affect normal locomotion. Vulval development and egg-laying behavior is affected by chd-3, egl-4, ina-1, and ser-2 and ser-4. Axonogenesis is enabled by ina-1, kal-1. and ptp-3. The mec genes here expressed as well affect interneurons implicated in touch sensation.

Table 43. Inter-neurons born at stage six and their genes.
New inter-neurons at this stage:
DVC PVDL PVDR PVR
New interneuron-specific genes:
cam-1 cav-1 cdc-42 ced-10 ceh-14 chd-3 deg-3 des-2 eat-4 egl-4 egl-46 fbxb-103 ggr-1 glr-1 glr-4 goa-1 hcp-3 ina-1 jkk-1 jnk-1 kal-1 lin-11 lin-35 lin-53 mec-10 mec-12 mec-3 mec-6 mec-7 mec-9 nhr-83 nlp-11 osm-9 pkc-1 ptp-3 sem-4 ser-2 ser-4 set-2 unc-103 unc-32 unc-40 unc-86

Table 44. **Description of interneuron specific genes at stage six.**

Gene	Description
Exc-7	exc-7 encodes an MRNA-binding protein required for the formation of the excretory canal
Deg-3, Des-2	deg-3 and des-2 are co-expressed and encode nicotinic acetylcholine receptors
Pkc-1	Pkc-1 encodes a serine/threonine kinase which affects acetylcholine release at neuromuscular junctions

Synapse Development in Stage Six

At stage six growth of the larvae is complete and all synapses are in place. At this point 2199 chemical synapses are in place. This number represents the non-redundant number of chemical synapses. Electrical synapses in this analysis have been omitted. 69 new synapses are formed at stage 6, with 48 of these being presynaptic, 20 are postsynaptic, and one is a new connection. The stage six, or final synaptic network may be viewed in the file "develstage6.txt" on the CD.

Summary of Biological Interpretation of Development Stage Data

In summary, this model predicts the following neural development. Stage one sees explosive growth in the neural system at 11 cell divisions. At this stage the amphid chemosensory neurons are born, and the ventral cord and ring motor neurons develop. This implies an early circuit between chemosensory neurons in the head and ventral body control. Stage two sees the birth of deirid sensory neurons and chemosensory neurons projecting into the ring. Notably in stage two there is the birth and migration of HSN, and the development of the vulval egg-laying apparatus. In stage three further mechanosensory neurons are born, and there is further development of interneurons innervating the ventral cord and ring. At stage four, more mechanosensory neurons are born, and motor neurons come into existence which innervate the dorsal and vulval muscles (VC series). Interneurons appear which implement reciprocal inhibition within the ventral cord, offering fine motor control. At stage five, dopaminergic neurons of the post-deirid sensillim are born, affecting feeding behavior. The VA and VB series of motor neurons are born, which innervate ventral muscles. There seems to be some development of thermotaxis at this stage. Finally, in stage 6, when the larval development of the

nervous system is nearly complete and there is the birth of a few interneurons that offer further locomotory control. There is some indication that innervation of the excretory canal is occurring at this late stage.

Summary of Developmental Model

In summary, a program has been developed that combines developmental, genetic, and neural information to indicate the complete neural development of *C. elegans*. A 280-neuron hermaphrodite was used as the model; the program divided the neurons and associated genes into six development stages. The program further generates a synaptic network for each development stage, providing completely novel information about the synaptic network development of *C. elegans*. A biological analysis was done on the output of the development stage program indicating the significant neurological and genetic events of each development stage.

Chapter 3

SYNAPTIC PATHWAY ANALYSIS

Introduction

Synaptic pathway analysis is possible due to a simple concept: that the nervous system of *C. elegans* may be represented as a graph. Here 'graph' is meant in the computer science sense of a data structure that has nodes and edges. The nodes represent neurons and the edges represent synapses. By representing the nervous system as a graph, graph programs may be used to navigate the nervous system. Many of the synaptic pathway programs use a routine called 'shortestpath,' which calculates the shortest path between two nodes in a graph, or, if the graph represents a nervous system, between two neurons in a synaptic network.

General Method

The nervous system is here represented, at any development stage, as a graph. The development stage program generates a graph file for each development stage which represents the synaptic network at that development stage. This development stage specific graph file is then fed into any one of the nine programs that have been developed to analyze the synaptic network. Each program performs a specific type of analysis on the synaptic network, which will be described in each of the in-silico experiments below.

It is a common question to ask, given a starting neuron, say a sensory neuron, and an ending neuron, say a motor neuron, what path or paths exist from one to the other? The program 'neuronpathstring' gives this information. This information is biologically useful as one may want to look at the nervous system from the standpoint of how inputs, such as sensory inputs,

are turned into motor outputs, and what circuitry is necessary to accomplish this. The same question may be asked of a set of input sensory neurons to a set of output motor neurons, i.e., where the set belongs to a sensory-motor subsystem, such as the chemosensory or mechanosensory subsystems. The program 'neuronpaths' accomplishes this. Also, one may want to analyze the nervous system to determine which inter-neurons form ganglia or which inter-neurons are high throughput inter-neurons with a large number of connections. The program 'subpaths' gives this information. One may want to know which neurons connect to neurons in a given neuronal path, this is accomplished by the program 'neuronenv.' Also, genetic information may be of interest. Genes associated with a specific neuronal path is given by 'pathgenes.' Genes associated with a set of neuronal pathways is given by 'pathsgenes.' Genes of neurons that impinge upon (have inputs to or outputs from) a given neuronal pathway are given by 'neuronenvgenes.' Genes associated with pairs of neurons along a pathway are given by the program 'pathgenespairs.' Partitioning of the synaptic network into 'strong components' is performed by 'strongcomponents.' A summary of the programs that have been developed for synaptic pathway analysis is given below.

Table 45. System of programs for analyzing the *C. elegans* nervous system.

Program	Function
Neuronpathstring	Gives the path through the synaptic network from an input neuron to an output neuron.
Neuronpaths	Gives the paths through the synaptic network from a set of input neurons (e.g. sensory) to a set of output neurons (e.g. motor).
Neuronenv	Outputs the neurons and synaptic network (the graph) of all neurons impinging upon a specified synaptic pathway.
Neuronenvgenes	Gives the genes associated with neurons which impinge upon a specified synaptic pathway.
Pathgenes	Gives the genes associated with a specific synaptic pathway.
Pathsgenes	Gives the genes associated with a set of synaptic pathways that occur between an input (sensory) and output (motor) set of neurons.
Subpaths	Gives the set of synaptic pathways that go through a set of neuronal subpaths. Useful for identifying ganglia and high throughput (high traffic) inter-neurons.
Pathgenespairs	Identifies genes associated with pairs of neurons along a synaptic pathway.
Strongcomponents	Partitions the synaptic network into 'strong components.'

Subpaths

The first and most exciting pathway analysis utility that has been developed is "subpaths." "Subpaths" identifies pathways that go through a specific neuron or small set of neurons, thereby identifying ganglia or other high throughput neurons. This utility is giving information about the structure of the *C. elegans* nervous system, not only identifying what synaptic pathways exist between an input set of neurons (e.g. sensory neurons) and an output set of neurons (e.g. motor neurons), but further analyzing these pathways to see where high traffic exists in the network, also thereby identifying critical high throughput neurons and subpaths.

Method: An example of how to invoke "Subpaths" under UNIX is as "./subpaths develstage6 sensory.txt motor.txt > outpufilename." Here "develstage6" is a graph file describing the total synaptic network of *C. elegans*, "sensory.txt" is a file describing a set of input sensory neurons, "motor.txt" is a file describing a set of output motor neurons, and "outputfilename" is the output file with the name of choice that contains the subpathway analysis. A sample of the output of this run is as follows:

Results:

Table 46. Paths through the subpath AVAL AS2.
Subpath is: AVAL AS2 Paths going through this subpath are:
ADEL AVAL AS2 DB1 ADER AVAL AS2 DB1 ADLL AVAL AS2 DB1 ASEL AIBL AVAL AS2 DB1 ASER AIBL AVAL AS2 DB1 ASGL AIBL AVAL AS2 DB1 ASHL AVAL AS2 DB1 ASHL AVAL AS2 DD1 ASIL AIBL AVAL AS2 DB1 ASJL ASKL AIBL AVAL AS2 DB1 ASKL AIBL AVAL AS2 DB1

The system identifies that 11 synaptic pathways pass through the subpath AVAL AS2. AVAL itself is a high-traffic neuron as is its contralateral counterpart AVAR. AVAL itself has 549 synaptic pathways that pass through it under this analysis. It is no surprise then that a number of pathways pass through the subpath AVAL AS2. AVAL is a ventral chord interneuron that synapses onto VA, DA, and, as here, AS motor neurons. AS2 is a ventral chord motor neuron that innervates dorsal muscles. DB1, the last neuron in many of these pathways acts as a negative inhibitor of motor neurons that innervate dorsal muscles. It is no surprise, then, that several pathways from sensory to motor neurons pass through the subpath AVAL AS2, as a the

stimulation of a number of sensory neurons, such as ASKL, ASJL, results in the innervation of dorsal muscles effected by AS2. If one were to ask which of these paths go through AIBL AVAL AS2, the answer would be:

Table 47. Paths through the subpath AIBL AVAL AS2
Subpath is: AIBL AVAL AS2 Paths going through this subpath are:
ASEL AIBL AVAL AS2 DB1 ASER AIBL AVAL AS2 DB1 ASGL AIBL AVAL AS2 DB1 ASIL AIBL AVAL AS2 DB1 ASJL ASKL AIBL AVAL AS2 DB1 ASKL AIBL AVAL AS2 DB1

AIBL is an amphid interneuron on the left side of the body. It is interesting to note the contralaterality here, as sensory neurons on the right side of the body, such as ASER pass through AIBL AVAL AS2, i.e., interneurons on the left side of the body. Six sensory neurons ultimately innervate dorsal motor neurons using the subpath AIBL AVAL AS2.

Discussion: Parsing through the information provided by the subpath analysis, the following high throughput neurons or subpaths were identified. AVAL had 549 connections, AIBL had 319 connections, AVAR, 294 connections, PVCL 187 connections, AIBR had 178 connections, HSNR had 120 connections, AIAL had 108 connections, etc… All of these neurons are identified by the program as high throughput neurons supporting a large number of incoming and outgoing connections, i.e., paths through them. Other interesting discoveries included neuron subpaths that act as relays between the left and right side of the nervous system (which is largely symmetrical). These include the subpaths ADAL AVAR, which had 63 connections, BDUR AVAL with 29 connections, RIFR PVCL with 27 connections, and ADLR PVCL also with 27 connections, and several others. Also discovered were some long subpaths

that had several connections running through them, such as AIBL AVAL PVCL VB8 with 6 paths running through it, AIAR ADLR PVCR VB8 with 6 paths running through it, AIAR ADLR PVCL with 27 paths running through it, AIBL AVAL PVCL with 24 paths running through it, and AIBL AVAL with 265 paths running through it as well as AIBR AVAR with 99 paths running through it. All of these high traffic neurons and subpaths are candidates for further study, as they have shown themselves to be critical for the operation of the nervous system.

Neuronpathstring

"Neuronpathstring" was run, and the output was analyzed for biological correctness.

Method: "Neuronpathstring" in UNIX is invoked as: "./neuronpathstring develstage6 ASKL VD10 > outputfile," where "neuronpathstring" is the name of the program, "develstage6" is the graph input file of the entire nervous system, and 'ASKL' is the starting neuron, here a single ciliated amphid chemosensory neuron, and "VD10" is a ventral chord motor neuron which innervates ventral body muscles. The inputs ask the program to provide the most direct path from ASKL to VD10 through the synaptic network of *C. elegans*.

Results: The "outputfile"indicates the following neuronal sequence: "ASKL AIBL AVAL PVCL VB8 VD10." This synaptic pathway of length 5 (5 synapses) includes the intermediary neurons "AIBL, AVAL, PVCL, and VB8."

Discussion: The chemosensory neuron ASKL synapses on AIBL which is an amphid inter-neuron, AIBL in turn synapses on AVAL which is a ventral chord inter-neuron which marks the transition of the pathway into the ventral region where it innervates muscles. AVAL synapses on PVCL which also is a ventral chord inter-neuron which is in the lumbar ganglion,

which in turn synapses on the VB8 motor interneuron which innervates ventral body muscles, but which also synapses on VD10 which is a reciprocal inhibitor of ventral chord motor neurons. Note that the beginning of the pathway, ASKL, AIBL, AVAL, and PVCL all end in 'L' indicating that they are on the left side of the worm. The program has found the pathway from the ASKL chemosensory neuron to the VD10 ventral motor neuron, which would be expected in the intact developing animal.

The synaptic pathway from any starting neuron to any ending neuron may be found. Typically one would look for a pathway between a sensory neuron and a motor neuron, but subpathways involving only inter-neurons may also be analyzed with the program. Here, also it should be noted this question is being asked in the context of the total mature nervous system (what "develstage6" describes), but pathway questions may be asked of the nervous system in development, by substituting in the graph file for any development phase of the nervous system.

Pathgenes

If the same question were asked, what is the pathway from ASKL to VD10, but what genes were associated with this pathway was of interest, the program "pathgenes" would be used. This program will give the genes associated with each neuron in the pathway, the union set of all genes involved in the pathway, and the intersection set of genes common to every neuron in the pathway.

Method: The program is invoked under UNIX as "./pathgenes NeuronGenesStage6 ASKL VD10 > outputfile." When 'outputfile' is opened, the program will indicate which genes are involved in this pathway. The file "NeuronGenesStage6" is a graph file like "develstage6"

describing the total synaptic network of *C. elegans*, but this file also maintains information on which genes are active in each neuron. The file is described in Appendix N, "Biological Data Sets." This is a standard file used in many of these analyses.

Results: The program yields the following output:

Table 48. Genetic analysis of the pathway from ASKL to VD10.
Path length is: 5 Path is: ASKL AIBL AVAL PVCL VB8 VD10
Neuron: ASKL has the following genes:
ahr-1 bra-1 cam-1 cav-1 cdc-42 ced-10 chd-3 che-3 daf-11 eat-4 egl-4 fbxb-103 gpa-14 gpa-15 gpa-3 ida-1 ina-1 kin-29 kvs-1 lin-35 lin-53 nlp-10 nlp-14 nlp-8 odr-1 osm-3 osm-6 osm-9 set-2 sgk-1 sra-7 sra-9 srg-2 srg-8 tax-2 tax-4 tax-6 unc-103 zig-4 zig-5
Neuron: AIBL has the following genes:
cav-1 cdc-42 ced-10 chd-3 fbxb-103 ggr-1 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 mgl-2 odr-2 ptp-3 set-2 tax-6 unc-8
Neuron: AVAL has the following genes:
cam-1 cav-1 cdc-42 ced-10 chd-3 csk-1 fbxb-103 flp-1 flp-18 ggr-2 glr-1 glr-2 glr-4 glr-5 gpa-14 ina-1 lin-11 lin-35 lin-53 nmr-1 nmr-2 pef-1 ptp-3 rig-3 sax-3 set-2 src-1 tax-6 unc-103 unc-42 unc-6 unc-8
Neuron: PVCL has the following genes:
ace-2 cam-1 cav-1 cdc-42 ced-10 cfi-1 chd-3 deg-1 deg-3 des-2 egl-3 egl-36 egl-5 glr-1 glr-2 glr-5 goa-1 gpa-16 ina-1 lin-35 lin-53 mab-21 mab-9 mec-6 nhr-83 nmr-1 nmr-2 rig-1 sax-3 ser-2 set-2 unc-8 vab-15
Neuron: VB8 has the following genes:
dbl-1 pag-3 unc-6
Neuron: VD10 has the following genes:
ced-10 npr-1 unc-25 unc-47
The genes that are common to every neuron in this pathway are:
There are no genes common to every neuron in this pathway according to the database.

Discussion: Now all of the genes recorded as active in this pathway are known. It is interesting to note that there a no genes which are common to every neuron in this path. It should be noted that several genes are repeated in the lists for different neurons. This is because the same gene may be expressed in many different neurons. The pathway is a pathway on the left side of the nervous system, where ASKL is an amphid chemosensory neuron which relays its information to AIBL, an amphid interneuron, which synapses on AVAL, a central switch that has 549 connections, which synapses on PVCL, a ventral cord interneuron found in the lumbar

ganglia which synapses on VB8, a ventral cord motor neuron, which is reciprocally inhibited by VD10. This circuit makes sense. It is a circuit from a chemosensory neuron, along the left side of the body, to a ventral motor neuron. The union set of genes found by the program is sensible as well.

Several genes here indicated affect the chemosensation of ASKL. Sng-2 encodes a seven transmembrane G-coupled chemosensory receptor expressed exclusively in the cilia of ASK neurons involved in the chemotaxis to lysine. Gpa-14 and 15 encode a G-protein alpha subunit of a heterotrimeric GTPase which is expressed in the ASK and also ASI, ASJ, and ASH chemosensory neurons. Gpa-3 enocdes the same and is involved in water soluble chemoattraction and aversion, and volatile chemoattraction. Eat-4 is a vesicular glutamate transporter which affects chemotaxis, thermotaxis, feeding and foraging behavior. Egl-4 is a cyclic GMP-dependent protein kinase which relays sensory cues that modulate chemosensory behavior involved in egg-laying and foraging. Kin-29 is a serine/threonine kinase involved in regulating the expression of chemosensory receptors. Kvs-1 is a voltage gated potassium channel involved in chemotaxis. Tax-6 encodes an ortholog of calcineurin, required for the inhibition and adaptation of several sensory neurons. And osm-6 and 9 encode a capsaicin receptor which mediates the response to some chemical stimuli and to ASH neuron mediated osmotic stimuli.

Genes here indicated which affect the motor component of this circuit are as follows. Affecting normal locomotion are pag-3, a zinc-finger protein expressed in ventral cord motor neurons, deg-3, a nicotinic acetylcholine receptor, and goa-1, a heterotrimeric G protein alpha subunit. Backward movement is affected by mab-21 and mab-9, a T-box transcriptional

regulator, and unc-42, a homeodomain protein required for axonal outgrowth in the backward command interneurons like AVAL. The sinusoidal movement characteristic of *C. elegans* is enabled by unc-8, an amiloride-sensitive cation-selective channel, and flp-1 and flp-18, an RMRF-amide related neuromodulatory peptide. The duration of forward movement is affected by nmr-1, an NMDA ionotropic glutamate receptor, and normal head movement is affected by mgl-2, a metabotropic glutamate receptor. Differentiation of PVCL is enabled by the DNA-binding protein cfi-1, and sax-3 is critical in confining migration of neuroblasts to specific ventral muscle quadrants.

Pathgenespairs

Introduction: However, a biologist asked if there are more genes common to neurons which synapse on eachother in a pathway than those that do not. This question can be answered using "pathgenespairs," or by simply running 'pathgenes' on contiguous and non-contiguous pairs of neurons in the pathway. Of biological interest here are genes which participate in synapse formation. Pathgenespairs works as follows. Once the pathway between two input neurons is found, genes common to contiguous and non-contiguous pairs are found. Suppose that the input neurons are A and E. And suppose pathgenespairs find the following pathway: ABCDE. Then pathgenespairs will also find the genes common to the contiguous pairs AB, BC, CD, and DE. Likewise, for comparison purposes, pathgenespairs will find the genes common to the non-contiguous pairs AC, AD, AE, BD, BE, and CE. These two sets may then be compared to verify whether contiguous neurons in a pathway have more genes in common than do non-contiguous genes. Of especial interest here are genes that participate in synapse formation.

Method: Pathgenespairs is invoked under UNIX as follows: "./pathgenespairs nrneurongenes ASKL VD10 > outputfile," where 'pathgenespairs' is the name of the program, 'nrneurongenes' is the name of the graph file with gene information included, here of the entire nervous system, and 'ASKL' is the input neuron, here a sensory neuron, and 'VD10' is the output neuron, here a motor neuron, and 'outputfile' is the name of the output file where you wish to place the results.

Results: The results are given below for the input neuron ASKL and output neuron VD10 calculated using the mature, develstage6 nervous system with genes per neuron represented.

Table 49. Genetic analysis of contiguous and non-continguous pairs of neurons in a pathway.
Begining neuron is: ASKL Ending neuron is: VD10 Path length is: 5 Path is: ASKL AIBL AVAL PVCL VB8 VD10
Genes common to CONTIGUOUS neurons in the path
The genes common to the contiguous pair ASKL AIBL are:
cav-1 cdc-42 ced-10 chd-3 fbxb-103 ina-1 lin-35 lin-53 set-2 tax-6
The genes common to the contiguous pair AIBL AVAL are:
cav-1 cdc-42 ced-10 chd-3 fbxb-103 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 ptp-3 set-2 tax-6 unc-8
The genes common to the contiguous pair AVAL PVCL are:
cam-1 cav-1 cdc-42 ced-10 chd-3 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 nmr-1 nmr-2 sax-3 set-2 unc-8
The genes common to the contiguous pair PVCL VB8 are: There are no genes common to these two neurons.
The genes common to the contiguous pair VB8 VD10 are: There are no genes common to these two neurons.
Genes common to NON-CONTIGUOUS neurons in the pathway:
The genes common to the non-contiguous pair ASKL AVAL are:
cam-1 cav-1 cdc-42 ced-10 chd-3 fbxb-103 gpa-14 ina-1 lin-35 lin-53 set-2 tax-6 unc-103
The genes common to the non-contiguous pair ASKL PVCL are:
cam-1 cav-1 cdc-42 ced-10 chd-3 ina-1 lin-35 lin-53 set-2
The genes common to the non-contiguous pair ASKL VB8 are: There are no genes common to these two neurons
The genes common to the non-contiguous pair ASKL VD10 are:
ced-10
The genes common to the non-contiguous pair AIBL PVCL are:
cav-1 cdc-42 ced-10 chd-3 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 set-2 unc-8
The genes common to the non-contiguous pair AIBL VB8 are: There are no genes common to these two neurons

The genes common to the non-contiguous pair AIBL VD10 are: ced-10
The genes common to the non-contiguous pair AVAL VB8 are: unc-6
The genes common to the non-contiguous pair AVAL VD10 are: ced-10
The genes common to the non-contiguous pair PVCL VD10 are: ced-10
The genes that are common to every neuron in this pathway are:
There are no genes common to every neuron in this pathway according to the database

Discussion: Genes common to the contiguous pair "ASKL AIBL" were "cav-1 cdc-42 ced-10 chd-3 fbxb-103 ina-1 lin-35 lin-53 set-2 tax-6." Genes common to the contiguous pair "AIBL AVAL" were "cav-1 cdc-42 ced-10 chd-3 fbxb-103 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 ptp-3 set-2 tax-6 unc-8." And the genes common to the contiguous pair "AVAL PVCL" were "cam-1 cav-1 cdc-42 ced-10 chd-3 glr-1 glr-2 glr-5 ina-1 lin-35 lin-53 nmr-1 nmr-2 sax-3 set-2 unc-8." But there were no genes common to the contiguous pair "PVCL VB8," or "VB8 VD10." Still this may be significant, as all neurons ending in 'L' on the left side of the nervous system, marking the transition from chemosensory to interneurons shared genes. The ventral motor neurons did not. Further analysis would be needed, and this analysis is subject to the gene information that exists per neuron as given by WormBase. Genes such as "glr" are glutamate receptor family genes, "nmr" NMDA glutamate ion channel receptor gene, and "sax-3," sensory axon guidance gene, as well as "unc-8," which expresses a mechanical ion channel primarily in the nerve ring, are important in nerve and neuronal synapse formation, as are the "cam" (cell adhesion molecule) genes. Further information on these genes and their associated proteins may be gained by using traditional bioinformatics approaches. Fewer genes were found to be common between non-contiguous pairs. The pairs ASKL, VB8 and AIBL, VB8 have no

genes in common. The pairs ASKL-VD10, AIBL-VD10, AVAL-VB8, AVAL-VD10, and PVCL-VD10 have only one gene in common, which in all cases is ced-10, an apoptosis gene, except for AVAL-VB8 where it is unc-6. In only three cases are several genes shared: ASKL-AVAL, ASKL-PVCL, and AIBL-PVCL.

Neuronenv

One may also be interested in the neurons and synapses that surround a particular pathway, or with the genes that are active in such surrounding neurons and synapses. This information may be attained by running the 'neuronenv' and 'neuronenvgenes' programs, respectively. 'Neuronenv' stands for 'neuron environment,' i.e., the environment of neurons and synapses that surround a particular pathway. 'Neuronenv' lists the neurons and synapses that impinge upon the neurons in a particular pathway. By 'impinge' is meant neurons that either have connections to or connections from neurons in the pathway.

Method: 'Neuronenv' may be invoked under UNIX by typing "neuronenv develstage6 ASKL VD10 outputfilename." This will find the same path as before: "ASKL AIBL AVAL PVCL VB8 VD10," but will also find all neurons and synapses that connect to this path. The output is a well-formed graph file, which includes all of the neurons and synapses that impinge upon the specified path. This graph file, in turn, can be input into any of the utilities that utilize a graph file as input, such as neuronpath, or strongcomponents.

Result: In this case the graph file generated includes 136 neurons and 208 synapses. The file is not here listed because of its size. The large number of neurons is due primarily to the fact that AVAL is in the pathway; AVAL is a high traffic neuron which has a high number of

incoming and outgoing connections, as the subpaths analysis has shown. A graph file is simply a text file that lists first the 136 neurons in the neuron environment, and then each of the 208 synapses in the format: neuron1 neuron1 weight, and example being "ADAL AIBL 1."

Discussion: The graph file produced by 'neuronenv' may be fed some of the other synaptic pathway analysis programs for further analysis. These programs include 'neuronpathstring,' 'neuronpaths,' 'subpaths,' and 'strongcomponents.' An example is feeding the output of neuronenv into neuronpathstring: "neuronpathstring neuronenvoutputfile ADEL AS2,' where 'neuronenvoutputfile' replaces the development stage graph, and 'ADEL,' and 'AS2' are input and output neurons in the 'neuronenvoutputfile' graph. 'Neuronenv' is thereby an effective tool to divide the nervous system into nervous system subparts centered around a synaptic pathway for further study, or an effective tool to generate subgraphs from graphs.

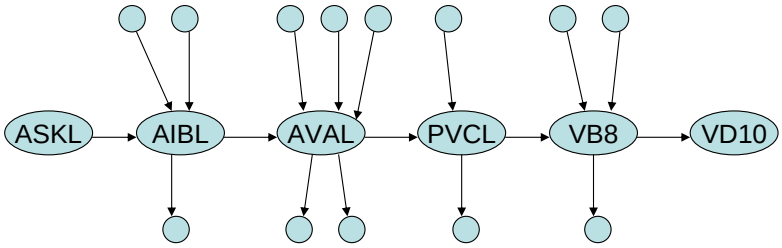

Figure 8. Sample neuron environment. A sample neuron environment consists of all neurons that have connections to or connections from a specified neuron path.

Neuronenvgenes

Introduction: The next utility specifies all genes associated with a neuron environment. This utility is called "neuronenvgenes." 'Neuronenvgenes' doesn't produce a graph file as does 'neuronenv,' but rather produces a list of each neuron in the neuron environment followed by the

genes associated with that neuron, a union list of all of the genes of the neuron environment, and the intersection of all of the genes in the neuron environment, if this is not the empty set.

Method: Neuronenvgenes is invoked under UNIX as "./neuronenvgenes nrneurongenes ASKL VD10 outputfile," as an example, where 'neuronenvgenes' is the name of the program, 'nrneurongenes' is a specially crafted graph file that includes genetic information per neuron, ASKL is the input neuron, VD10 is the output neuron, and 'outputfile' is the name of the file where the results are to be placed.

Result: The output includes genetic information on each neuron in the neuron environment, the union of such genes, and the intersection of such genes. Neuron environments may be small or large depending upon the number of connections to neurons in the path. Neuron environments are one way to carve up the nervous system to analyze it, for further pathway or genetic analysis.

Table 50. Output of neuronenvgenes. The output indicates the genes associated with a neuron environment.
Begining neuron is: ASKL Ending neuron is: VD10 Path length is: 5 Path is: ASKL AIBL AVAL PVCL VB8 VD10
Genes involved in the neuron circuit that stems from the pathway: ASKL AIBL AVAL PVCL VB8 VD10 are:
ace-2 ace-3 ahr-1 avr-15 bra-1 cam-1 cat-1 cat-2 cav-1 cdc-42 cdh-3 ced-10 ceh-10 ceh-13 ceh-14 ceh-17 ceh-23 ceh-24 ceh-32 ceh-36 ceh-43 ceh-6 cfi-1 cha-1 chd-3 che-1 che-2 che-3 cki-1 clh-3 cnb-1 cog-1 csk-1 daf-1 daf-11 daf-14 daf-28 daf-4 daf-5 daf-7 dat-1 dbl-1 deg-1 deg-3 des-2 dop-1 dop-2 dyn-1 eat-16 eat-20 eat-4 egl-2 egl-21 egl-3 egl-36 egl-4 egl-43 egl-44 egl-46 egl-5 exc-7 fax-1 fbxb-103 flp-1 flp-18 flp-21 flr-4 flt-1 gar-1 gar-2 gcs-1 gcy-32 gcy-33 gcy-34 gcy-35 gcy-36 gcy-37 gcy-5 gcy-6 gcy-7 ggr-1 ggr-2 glr-1 glr-2 glr-4 glr-5 glr-8 gly-18 goa-1 gpa-1 gpa-10 gpa-11 gpa-13 gpa-14 gpa-15 gpa-16 gpa-2 gpa-3 gpa-4 gpa-5 gpa-6 gpa-8 gpa-9 gpb-2 gpc-1 grd-6 grk-2 gsa-1 ham-2 hbl-1 hcp-3 hen-1 hid-1 hlh-14 hlh-2 hmr-1 ida-1 igcm-2 ina-1 inx-4 itr-1 jkk-1 jnk-1 kal-1 kin-29 kvs-1 lad-2 let-60 lim-4 lim-6 lin-11 lin-14 lin-29 lin-35 lin-53 lst-1 lsy-6 mab-21 mab-23 mab-9 mec-10 mec-2 mec-3 mec-4 mec-6 mec-7 mec-8 mec-9 mgl-1 mgl-2 mig-1 mig-13 mig-2 mps-1 mtd-1 ncs-1 nhr-22 nhr-79 nhr-83 nhx-5 nid-1 nlp-1 nlp-10 nlp-11 nlp-13 nlp-14 nlp-15 nlp-18 nlp-24 nlp-27 nlp-3 nlp-5 nlp-6 nlp-7 nlp-8 nlp-9 nmr-1 nmr-2 npr-1 nsy-1 ocr-1 ocr-2 odr-1 odr-2 odr-3 oig-3 opt-3 osm-10 osm-3 osm-6 osm-9 pag-3 pat-4 pef-1 pkc-1 ptl-1 ptp-3 qui-1 rgs-2 rhgf-1 rig-1 rig-3 sax-3 sax-7 scc-1 sem-4 ser-2 ser-4 set-2 sgk-1 skn-1 slt-1 sra-10 sra-11 sra-13 sra-6 sra-7 sra-9 srb-6 src-1 srd-1 sre-1 srg-2 srg-8 srh-220 sro-1 srp-2 str-2 str-3 syg-1 tag-97 tax-2 tax-4 tax-6 tba-1 tba-2 tbx-2 tol-1 tph-1 trp-1 ttx-3 unc-103 unc-129 unc-14 unc-17 unc-2 unc-25 unc-3

unc-30 unc-32 unc-37 unc-4 unc-40 unc-42 unc-47 unc-5 unc-51 unc-53 unc-6 unc-71 unc-73 unc-76 unc-8 unc-86 unc-97 vab-15 vab-7 ver-2 ver-3 wrk-1 zig-1 zig-2 zig-3 zig-4 zig-5 zig-8

First each neuron in the neuron environment and the genes associated with each neuron are given. Then the union and intersection of all genes in the neuron environment are given. The output has been here abbreviated, as there are 136 neuron total in this neuron environment. 'Neuronenvgenes' also does not generate the synapses for the neuron environment; for this use 'neuronenv.'

Discussion: The use of 'neuronenvgenes' is to determine the genetic makeup of a neuron environment. In this way, the genes active in this slice of the nervous system may be investigated. The neuron environment in this case is large due to the fact that AVAL is in the pathway, and AVAL has a large number of incoming and outgoing connections. AVAL is a major switching point in the *C. elegans* nervous system. In this case 'neuronenvgenes' was run on the mature nervous system, but the same utility will work on slices of the developing nervous system as well, if an earlier stage graph is substituted for 'nrneurongenes' in the invocation.

Neuronpaths

Introduction: The next two utilities work on multiple pathways, i.e., all pathways that exist between a set of input neurons (e.g. sensory neurons) and a set of output neurons (e.g. motor neurons). The first utility is "neuronpaths." "Neuronpaths" simply gives all paths from a set of input neurons to a set of output neurons. This allows analysis of specific subsystems such as the chemosensory subsystem or mechanosensory subsystem. Here one would merely input a file with the names of the sensory neurons participating in the chemosensory subsystem and an

input file with the names of the motor neurons associated with a specific motor subsystem, such as the ventral chord motor neurons, and 'neuronpaths' would indicate the pathways between all of the chemosensory neurons and the ventral chord motor neurons.

Method: The utility is invoked as "./neuronpaths develstage6 sensory.txt motor.txt outputfilename" under UNIX, where 'develstage6' is a graph file representing the total *C. elegans* synaptic network, 'sensory.txt' is the file that contains the names of chemosensory neurons, 'motor.txt' is a file that contains the names of ventral chord motor neurons, and 'outputfilename' is the name of the file to which one wants the output sent.

Results: The output is a series of paths that looks something like the following:

Table 51. Output of neuronpaths (abbreviated).
Path length is: 2 Path is: ADEL AVAR AS1
Path length is: 2 Path is: ADEL AVAL AS2
Path length is: 2 Path is: ADEL AVAL AS3
. . .
Path length is: 4 Path is: ASKR RIFR PVCL VB8 VD10
Path length is: 3 Path is: ASKR AIMR AVFR VD11
Path length is: 3 Path is: ASKR AIMR PVNR VD12
Path length is: 4 Path is: ASKR AIAR ADLR AVAR

Discussion: Note that path length is the number of synapses involved. Here the listing has been abbreviated as it is otherwise too long. The program generates a path from every input neuron in the input set to every output neuron in the output set. If each input set has 10 members, for example, 100 pathways will be generated. In this way the path from any input (e.g. sensory) neuron to any output (e.g. motor) neuron may be examined for further study. Each path

indicates the synaptic pathway that is involved in converting the sensory input into the motor output. Collectively the set of synaptic pathways from the input (sensory) neurons to the output (motor) neurons defines the circuit or subgraph for that part of the nervous system. By carefully selecting the input neurons as, say, from the set of amphic chemosensory neurons, and the motor neurons as, say, the ventral cord motor neurons, a particular circuit may be studies in terms of its synaptic network.

Pathsgenes

Introduction: The second utility is "pathsgenes" that's 'paths' with an 's.' This utility does the same thing as 'pathgenes' except it works on multiple pathways, again, all of the pathways that exist between an input set of neurons, such as sensory neurons and an output set of neurons, such as motor neurons. For every path generated there is given the union set of genes involved in the pathway and the intersection set (genes common to every neuron in the pathway), and the union set and intersection set of all genes in all pathways calculated is given as well.

Method: 'Pathsgenes' is invoked under UNIX as: "./pathsgenes nrneurongenes sensory.txt motor.txt > outputfilename," where 'pathsgenes' is the name of the program, 'nrneurongenes' is a standard file containing information about all *C. elegans* neurons, and their genes (as well as lineage information), as well as the synaptic connections of the mature worm, and 'sensory.txt' is the set of sensory input neurons of interest in the study, and 'motor.txt' is the set of motor output neurons of interest, and 'outputfilename' is the filename of choice where the results are stored.

Results: For each synaptic pathway identified the output below is generated. The number of synaptic pathways listed is the number of input (sensory) neurons times the number of output (motor) neurons. The output generated by this program is abbreviated for space reasons (the entire output is on the CD in the file entitled ‘pathsgenes-output.txt).

Table 52. Output of pathsgenes.
Begining neuron is: ADEL Ending neuron is: AS1 Path length is: 2 Path is: ADEL AVAR AS1
Genes involved in pathway: ADEL AVAR AS1 are: cam-1 cat-2 cav-1 cdc-42 ced-10 chd-3 che-2 che-3 csk-1 dat-1 dop-2 egl-36 fbxb-103 flp-1 flp-18 ggr-2 glr-1 glr-2 glr-4 glr-5 gpa-14 hmr-1 ida-1 ina-1 lin-11 lin-35 lin-53 nmr-1 nmr-2 osm-6 pef-1 ptp-3 rig-3 sax-3 set-2 src-1 tax-6 unc-103 unc-42 unc-53 unc-6 unc-8
Begining neuron is: ADEL Ending neuron is: AS2 Path length is: 2 Path is: ADEL AVAL AS2
Genes involved in pathway: ADEL AVAL AS2 are: cam-1 cat-2 cav-1 cdc-42 ced-10 chd-3 che-2 che-3 csk-1 dat-1 dop-2 egl-36 exc-7 fbxb-103 flp-1 flp-18 ggr-2 glr-1 glr-2 glr-4 glr-5 gpa-14 hmr-1 ida-1 ina-1 lin-11 lin-35 lin-53 nmr-1 nmr-2 osm-6 pef-1 ptp-3 rig-3 sax-3 set-2 src-1 tax-6 unc-103 unc-42 unc-53 unc-6 unc-8
. . .
Begining neuron is: ASKR Ending neuron is: VD13 Path length is: 4 Path is: ASKR AIAR ADLR AVAR VD13
Genes involved in pathway: ASKR AIAR ADLR AVAR VD13 are: ahr-1 bra-1 cam-1 cav-1 cdc-42 ced-10 ceh-23 ceh-32 chd-3 che-3 cog-1 csk-1 daf-11 eat-4 egl-4 fbxb-103 flp-1 flp-18 flp-21 ggr-2 glr-1 glr-2 glr-4 glr-5 gpa-1 gpa-11 gpa-14 gpa-15 gpa-2 gpa-3 gpc-1 hlh-2 ida-1 ina-1 kin-29 kvs-1 lin-11 lin-35 lin-53 mgl-1 nhr-79 nlp-10 nlp-14 nlp-7 nlp-8 nmr-1 nmr-2 npr-1 ocr-1 ocr-2 odr-1 osm-3 osm-6 osm-9 pef-1 ptp-3 qui-1 rig-3 sax-3 set-2 sgk-1 sra-7 sra-9 srb-6 src-1 sre-1 srg-2 srg-8 srh-220 sro-1 tax-2 tax-4 tax-6 ttx-3 unc-103 unc-25 unc-42 unc-47 unc-6 unc-8 ver-2 zig-4 zig-5

Discussion: 'Pathsgenes' generates information about the pathways from a set of input, sensory neurons to a set of output, motor neurons, and the genes associated with each of those pathways. Here, for example, the pathway from ADEL to AS1 involves 3 neurons and 2 synaptic connections, and the pathway from ASKR to VD13 involves 5 neurons and 4 synaptic connections, and the genes associate with each of these pathways is given. Of interest here is the genetics of the circuit, i.e., the genetics of all of the pathways found between the sensory input set and motor output set of neurons. This information is generated by the program, but too voluminous to be presented here.

StrongComponents

Next, ‘strongcomponents’ was run. The graph of the synapses of the entire *C. elegans* nervous system was partitioned into ‘strong components.’

Description: A ‘strong component’ is a portion of the graph or of the synaptic network that is particularly strongly connected. More precisely, every node within the strong component subnetwork is reachable from every other node and conversely. So if A and P are nodes within the strong component subnetwork, there exists a path from A to P and a path from P to A. There exists an algorithm which partitions graphs into their strong components. This algorithm was run on the entire synaptic network of *C. elegans* to see which groups of neurons formed strong components.

Method: A new application, called ‘strongcomponent’ was run under UNIX with the following parameters: “strongcomponent nrsynapses.txt StrongComp,” where ‘strongcomponent’ is the UNIX name of the application, ‘nrsynapses.txt’ contains the list of all neurons and synapses (edges) of the entire *C. elegans* nervous system, and ‘StrongComp’ is the output file which contains the list of strong components for the entire nervous system.

Results: The results are contained in the output file ‘StrongComp,’ which is visible in Appendix B, Biological Data Sets, here listed as well.

Table 53. Output of Strongcomponents.
Strong Component 1: VC6
Strong Component 2: SDQR
Strong Component 3: PVDR
Strong Component 4: PLNR
Strong Component 5: PLML
Strong Component 6: PHCR
Strong Component 7: IL2DR
Strong Component 8: IL2DL
Strong Component 9: DVB
Strong Component 10: PHCL
Strong Component 11: ASIR
Strong Component 12: ASIL
Strong Component 13: AINL
Strong Component 14: ADAL ADAR ADEL ADER ADFL ADFR ADLL ADLR AFDL AFDR AIAL AIAR AIBL AIBR AIML AIMR AINR AIYL AIYR AIZL AIZR ALA ALML ALMR ALNL ALNR AQR AS1 AS11 AS2 AS3 AS4 AS5 AS6 AS9 ASEL ASER ASGL ASGR ASHL ASHR ASJL ASJR ASKL ASKR AUAL AUAR AVAL AVAR AVBL AVBR AVDL AVDR AVEL AVER AVFL AVFR AVG AVHL AVHR AVJL AVJR AVKL AVKR AVL AVM AWAL AWAR AWBL AWBR AWCL AWCR BAGL BAGR BDUL BDUR CEPDL CEPDR CEPVL CEPVR DA1 DA2 DA3 DA4 DA5 DA6 DA9 DB1 DB2 DB3 DB4 DB7 DD1 DD2 DD5 DVA DVC FLPL FLPR HSNL HSNR IL1DL IL1DR IL1L IL1R IL1VL IL1VR IL2L IL2R IL2VL IL2VR LUAL LUAR OLLL OLLR OLQDL OLQDR OLQVL OLQVR PDA PDB PDEL PDER PHAL PHAR PHBL PHBR PLMR PLNL PQR PVCL PVCR PVDL PVM PVNL PVNR PVPL PVPR PVQL PVQR PVR PVT PVWL PVWR RIAL RIAR RIBL RIBR RICL RICR RID RIFL RIFR RIGL RIGR RIH RIML RIMR RIPL RIPR RIR RIS RIVL RIVR RMDDR RMDL RMDR RMDVL RMED RMEV RMFL RMFR RMGL RMGR RMHL RMHR SAADL SAADR SAAVL SAAVR SABD SDQL SMBDL SMBDR SMBVL SMBVR SMDDL SMDDR SMDVL SMDVR URADL URADR URAVL URAVR URBL URBR URXL URXR URYDL URYDR URYVL URYVR VA1 VA11 VA12 VA2 VA3 VA4 VA5 VA6 VA8 VA9 VB1 VB10 VB11 VB2 VB3 VB4 VB5 VB6 VB8 VB9 VC1 VC2 VC3 VC4 VC5 VD1 VD10 VD11 VD12 VD13 VD2 VD3 VD5 VD6 VD8
Strong Component 15: DA7
Strong Component 16: AS8
Strong Component 17: AS7
Strong Component 18: VA10
Strong Component 19: VA7
Strong Component 20: VB7
Strong Component 21: SIADL
Strong Component 22: VD7
Strong Component 23: VD9
Strong Component 24: DD4
Strong Component 25: DB6
Strong Component 26: DB5
Strong Component 27: SIADR
Strong Component 28: SIBDL
Strong Component 29: SIBVL
Strong Component 30: SIAVL
Strong Component 31: SIAVR
Strong Component 32: RMEL

Strong Component 33: RMER
Strong Component 34: RMDDL RMDVR
Strong Component 31: SIAVR Strong Component 32: RMEL Strong Component 35: SIBVR
Strong Component 36: SIBDR
Strong Component 37: DA8
Strong Component 38: SABVR
Strong Component 39: SABVL
Strong Component 40: DD6
Strong Component 41: AS10
Strong Component 42: DD3
Strong Component 43: VD4

Discussion: The strong component analysis showed one large strong component, i.e., one large set of neurons which have mutually accessible paths to eachother. 248 out of 280 neurons, or 88.57% of the nervous system is part of this one large strong component. What this indicates biologically is that most neurons in the *C. elegans* nervous system have a synaptic path to most other neurons, i.e., it is highly interconnected with a high degree of feedback in this respect. Consider the following diagram to understand the nature of strong components and how a single vertex may be a strong component.

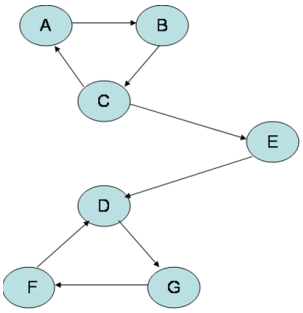

Figure 9. Strong Components Graph.

In the graph above, there are three strong components: {A,B,C}, {D,F,G} and {E}. {A,B,C} is a strong component as every member in this group has a pathway to every other member. A has a pathway to C through B and C has a pathway to B through A. Therefore A and C are mutually accessible and belong to the same strong component. B has a pathway to B has a pathway to A through C, and A has a direct pathway to B, so A and B are mutually accessible. Likewise, C and B are mutually accessible. The same is true of the group {D,F,G}. But E is a single vertex which is a strong component. This is because E stands on its own. E is

not mutually accessible with any other vertex. This is so as E has no pathway back to any vertex that points to E and no vertex has a pathway back to E that E points to. It could be said that E is a feedforward vertex but that the other vertices in the other strong components participate in feedback loops. Now, running 'strongcomponents' on the *C. elegans* nervous system identified one large strong component where every neuron in the strong component is mutually accessible with every other neuron through some pathway, and it also identified 43 neurons which are single vertex strong components, i.e., exist in the network in a feedforward manner with no mutual accessibility to all of the other neurons in the one large strong component. These neurons should be of especial neuroscientific interest, as they are feedforward points in the information processing of the *C. elegans* nervous system.

Table 54. Sensory feedback loops.
5 Sensory neurons (12.5% of sensory) with no feedback loop (entry only):
PLML, ASIR, ASIL, PHCL, PHCR
35 sensory neurons (87.5% of sensory) have a feedback loop:
ADEL ADER ADFL ADFR ADLL ADLR AFDL AFDR ALML ALMR ASEL ASER ASGL ASGR ASHL ASHR ASJL ASJR ASKL ASKR AVM AWAL AWAR AWBL AWBR AWCL AWCR PDEL PDER PHAL PHAR PHBL PHBR PHCL PHCR PLMR PVM

If further analysis is done, it is interesting to discover that only 5 out of 35 sensory neurons are entry-only neurons, i.e., have no feedback to them. The other 35 neurons, or 87.5% of sensory neurons, have other neurons which form feedback loops with them. This was not entirely expected. Note that in some cases which neurons have feedback or not is asymmetric, as PLML has no feedback loop, yet PLMR does.

Table 55. Motor feedback loops.
21 motor neurons (20%) have no feedback loop (exit only):
VC6 DA7 AS8 AS7 VA10 VA7 VB7 VD7 VD9 DD4 DB6 DB5 RMEL RMER DA8 DD6 AS10 DD3 VD4
84 motor neurons (80%) have a feedback loop.
AS1 AS2 AS3 AS4 AS5 AS6 AS9 AS11 DA1 DA2 DA3 DA4 DA5 DA6 DA9 DB1 DB2 DB3 DB4 DB7

DD1 DD2 DD5 HSNL HSNR OLLL OLLR OLQDL OLQDR OLQVL OLQVR PDA PDB RIML RIMR RMDDL RMDDR RMDL RMDR RMDVL RMDVR RMED RMEV RMFL RMFR RMHL RMHR URADL URADR URAVL URAVR VA1 VA2 VA3 VA4 VA5 VA6 VA8 VA9 VA11 VA12 VB1 VB2 VB3 VB4 VB5 VB6 VB8 VB9 VB10 VB11 VC1 VC2 VC3 VC4 VC5 VD1 VD2 VD3 VD5 VD6 VD8 VD10 VD11 VD12 VD13

21 motor neurons are exit-only, i.e, do not synapse on other neurons as a feedback loop, where as most, 80% of motor neurons do have some sort of feedback loop to the rest of the synaptic network. This implies that in most cases the motor output is modified by feedback loops. Note that which motor neurons have feedback loops is not symmetric, i.e., all members of a series, e.g. all VC, DA, VA, VB, DD, AS, or VD neurons do not all have either feedback or no feedback. The design of the *C. elegans* nervous system is selective in which motor neurons have feedback loops.

Table 56. Inter-neuron feedback loops.
16 interneurons (12%) are feedforward only:
SDQR PLNR DVB AINL SIADL SIADR SIBDL SIBVL SIAVL SIAVR SIBVR SIBDR SABVR SABVL IL2DL IL2DR
119 interneurons (88%) are involved in feedback loops:
ADAL ADAR AIAL AIAR AIBL AIBR AIML AIMR AINR AIYL AIYR AIZL AIZR ALA ALNL ALNR AQR AUAL AUAR AVAL AVAR AVBL AVBR AVDL AVDR AVEL AVER AVFL AVFR AVG AVHL AVHR AVJL AVJR AVKL AVKR AVL BAGL BAGR BDUL BDUR CEPDL CEPDR CEPVL CEPVR DVA DVC FLPL FLPR IL1DL IL1DR IL1L IL1R IL1VL IL1VR IL2L IL2R IL2VL IL2VR LUAL LUAR PLNL PQR PVCL PVCR PVDL PVDR PVNL PVNR PVPL PVPR PVQL PVQR PVR PVT PVWL PVWR RIAL RIAR RIBL RIBR RICL RICR RID RIFL RIFR RIGL RIGR RIH RIPL RIPR RIR RIS RIVL RIVR RMGL RMGR SAADL SAADR SAAVL SAAVR SABD SDQL SMBDL SMBDR SMBVL SMBVR SMDDL SMDDR SMDVL SMDVR URBL URBR URXL URXR URYDL URYDR URYVL URYVR

Again, as might be expected, most interneurons are involved in feedback loops and are part of the one large strong component, however, some, 12%, are feedforward only, i.e., they are their own strong component, as it was explained that a single vertex may be. It may be that these feedforward interneurons are at the periphery of the single large strong component synaptic network.

The strong component analysis elucidates some facts that one might not otherwise expect about the structure of the *C. elegans* nervous system. Most sensory neurons are not entrance only neurons but have feedbacks to them; most motor neurons are not exit only neurons but provide feedback to the rest of the nervous system as well. 88.57% of the nervous system is composed of a highly interconnected synaptic network where each neuron has a path to every other neuron in the synaptic network, i.e., is part of a single large strong component. Feedback mechanisms are used extensively in the design of the *C. elegans* nervous system.

Summary

In summary, nine utilities have been presented which allow further analysis of the *C. elegans* nervous system. These utilities are possible because the *C. elegans* nervous system may be represented as a graph, and hence graph utilities may be run on the representation of the nervous system. Specifically, these utilities allow finding the pathway through the nervous system from a starting (sensory) neuron to a final (motor) neuron, finding the set of pathways between a set of input (sensory) neurons and output (motor) neurons to identify pathways involved in a sensori-motor circuit; the inputs and outputs to a specific synaptic pathway may be found, and their associated genes. The genes involved in a specific pathway or a set of pathways between input and output neurons may be found. Significantly, an analysis that identifies high throughput ganglia and other high throughput neurons in the *C. elegans* nervous system is also described. The intersection of genes of contiguous and non-contiguous neurons in a pathway are compared to determine if more genes are involved in contiguous pairs, and if those genes are, for example, involved in synaptic formation. Finally, the graph describing the *C. elegans* nervous system is

partitioned into sets of neurons which have a unique integrity, that is, every neuron in the group has a pathway to every other neuron in the group. All of these utilities may be run on any development stage of the nervous system, and several utilities produce output which may be analyzed by other utilities. Together these utilities provide a set of tools to analyze the *C. elegans* nervous system, particularly its synaptic network.

Chapter 4

CORRELATIONS BETWEEN DEVELOPMENT AND SYNAPSE FORMATION

Next, there was an attempt to find numerical relationships between lineage distance and synaptic distance. By 'lineage distance' is meant the number of cell divisions by which the development of two neurons differ; this was indicated by comparing lineage strings and counting the number of characters after which a difference between the two strings is first detected. By 'synaptic distance' is meant the graph distance between two neurons in the synaptic network; this is equal to the number of synapses between the two neurons in the synaptic network. By comparing lineage distance and synaptic distance, the question is being raised whether a difference in DEVELOPMENT of the neuron has an impact in SYNAPSE FORMATION.

First, two applications were written. One measured the lineage distance between pairs of neurons; this was named 'lindist.' The other measured the synaptic (graph) distance between pairs of neurons; this was named 'syndist.' 'Syndist' used the shortestpath algorithm to measure the shortest graph distance between two neurons. It was important in order to compare these two quantities that the exact same pairs of neurons were measured in exactly the same order. In this study, a set of 280 neurons is used. By comparing the distance of every neuron to every other neuron, a set of 280 x 280, or 78,400 distances were measured in each case. Because of the computational intensity of such an algorithm, the programs were written in C and C++ respectively, and run on a 64-bit computer. Once the distances were calculated, a correlation between the two 78,400 cell long vectors was performed. The results are presented below.

Before presenting the results below, consider the computing platform in which the analyses were performed. The Biobike computing platform, hosted by Stanford, and found at http://www.biobike.org , was used to invoke syndist, lindist, and the application which performed the correlation between these distances. A diagram of the platform is given below:

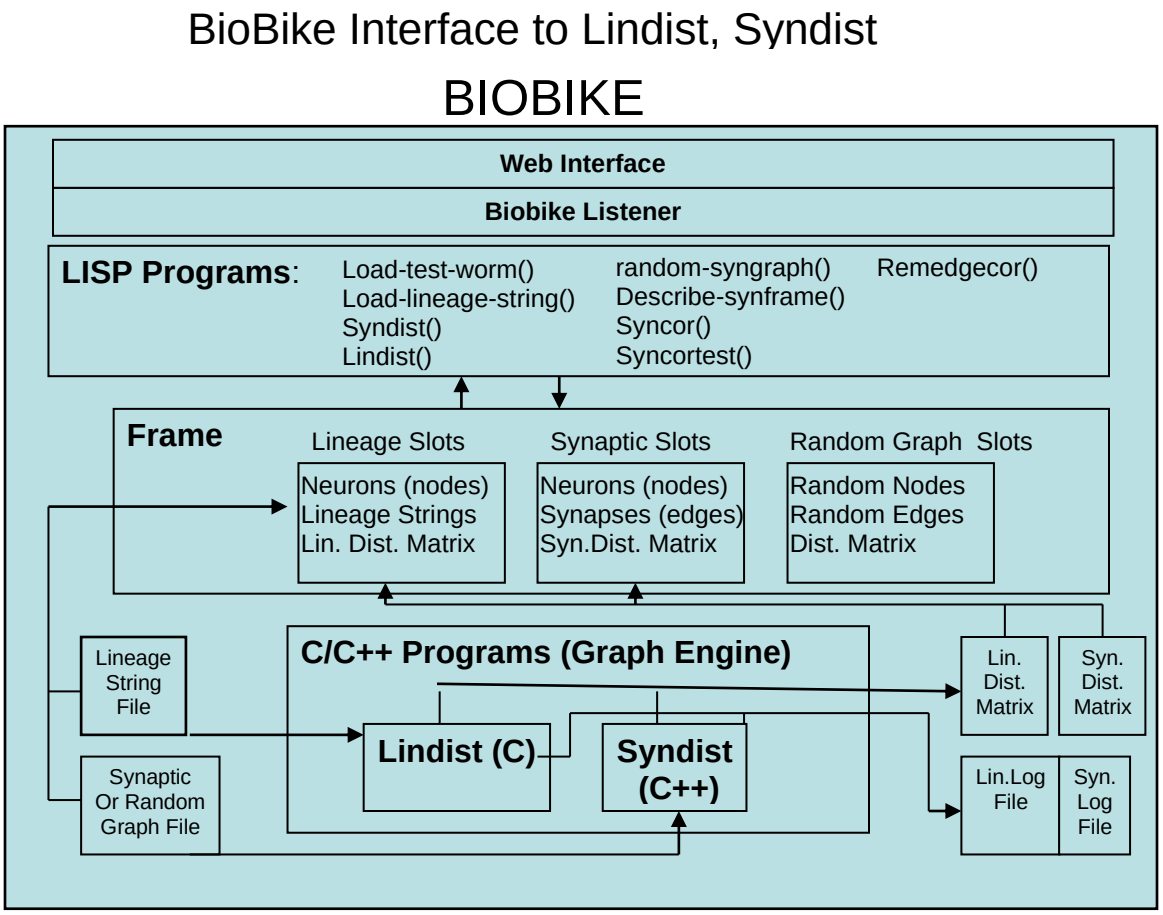

Figure 10. The Biobike Interface to Lindist and Syndist.

Through a web interface, the lindist and syndist programs are invoked with file arguments, in the case of lindist the file contains the lineage strings associated with each of the 280 neurons; in the case of syndist, the input file consists of a graph with a list of the 280

neurons and their synapses. The programs are run through the biobike web interface and the numerical distance results are stored in matrices. A biobike LISP application 'fcor' is then run to run a correlation between the lineage distance matrix and the synaptic distance matrix. In the summary of this paper, more will be said about biobike, which represents the future of this computing project.

In-Silico Experiment One:

Description: First the lineage distance and synaptic distance for the entire worm nervous system (development stage 6) was computed, and correlated.

Method:

1. First the lineage string file for the entire *C. elegans* nervous system (File N. of the Data Appendix) was loaded into a frame of Biobike, with the command: "(load-lineage-string :file "/home/ericw/c/worm/lindist/lineage280.txt")."
2. Next the lineage distance matrix was calculated and stored in the same frame with the command: "(lindist frameM)" where 'frameM' refers to the frame created in step 1 above. This created a 280 x 280 matrix of lineage distances in frameM.
3. Next the synaptic graph file, nodes and edges, was loaded into a different frame with the command: "(load-test-worm :file "/home/ericw/c/worm/syndist/Syn2.txt")." This created frameN.
4. Next the synaptic (or graph) distance was calculated between each node and every other node, resulting in a 280 x 280 matrix of synaptic distances, stored in frameN, with the

command: “(syndist frameN).”

5. Finally, the two matrices, the lineage and the synaptic matrices, were correlated with the ‘fcor’ command: “(fcor frameM frameN).”

Results: The following was the output for the correlation.

<9>> (fcor #$Temp9 #$Temp11)
> :: (:R 0.011312108 :R2 1.2796378e-4)

(5) Correlation of synaptic distance and lineage distance of the entire organism.

Discussion: The results are somewhat disappointing, since the correlation, approximately zero, indicates no strong positive or negative correlation between lineage and synaptic distance. But the experiment had to be performed to know that there is no strong positive or negative correlation between lineage and synaptic distance. Explanations for this are myriad. Many neurons migrate during neurogenesis so that the position indicated by the lineage string is not fixed, and hence is variable compared to the fixed nature of the synaptic network in the final animal.

Experiment Two: Lineage and Synaptic distances of the Left Side of the nervous system were compared.

Description: It was reasoned that a specific side or subsystem of the nervous system may show more correlation between lineage and final synaptic position.

Method:

1. First, a file containing just the lineage strings corresponding to the left side of the nervous system was generated. A list of the neurons corresponding to the left side (Data file N.), composed of 93 neurons (typically neurons ending in 'L' for left side, such as 'AVEL' and 'SAADL'), was fed into a program into which was also input a list of all neurons and their lineage strings (Data file N. "lineage280.txt"). Running the program "./newlineage LeftNeurons.txt lineage280.txt LeftLineage" under UNIX produced a text file 'LeftLineage' which contained only the 93 lineage strings of the left side of the developing nervous system.
2. Next, the left lineage strings were loaded into a frame on Biobike. This was accomplished with the command: "(load-lineage-string :file "/home/ericw/c/worm/lindist/LeftLineage")." This created a frameP slot with 93 left neurons and their corresponding lineage strings.
3. Next, the lineage distance matrix was calculated for the left side of the nervous system with the command "(lindist FrameP)," where the resulting 93 x 93 matrix was stored in FrameP.
4. Next, the synaptic graph corresponding to the left side of the nervous system had to be generated. Since this was a subset of the graph of the entire nervous system, a program was written to take the list of left neurons, and the list of edges (synapses) of the entire nervous system, and generate from this the list of edges (synapses) that only refer to left neural nodes. This was accomplished with the command: "./newedges LeftNeurons.txt

AllEdges.txt LeftEdges," under UNIX, where 'LeftNeurons.txt' refers to Data File N (of the Appendix), and 'AllEdges.txt' refers to the set of synapses of the entire nervous system (Data File N of the Appendix), and 'LeftEdges' is a text file which contains just the 840 synapses of the left side of the nervous system.

5. Next, the 93 neurons (nodes) and 840 synapses (edges) of the left side of the nervous system were loaded into a BioBike frame FrameQ.
6. Next, the synaptic distance matrix was calculated for the left side of the nervous system. The command "(syndist :File "/home/ericw/c/worm/syndist/LeftEdges")" was run from Biobike, generating a 93 x 93 matrix of synaptic (graph) distances, comparing each left neuron to every other left neuron.
7. Finally, a correlation between the lineage distance and synaptic distance of the left side of the nervous system was performed with the BioBike command "(fcor FrameP FrameQ)."

Results:

Left Side Correlation of Left Lineage Distance Matrix with Left Synaptic Distance.

Matrix:

> <22>> (fcor #$Temp14 #$Temp16)
> :: (:R 0.05777238 :R2 0.0033376478)

(6) Synaptic-lineage correlation of the left side of the organism.

Discussion: Again, the correlation of lineage distance with synaptic distance for the left side of

the nervous system is approximately zero, indicating no strong positive or negative correlation between these two matrices.

Experiment Three: Lineage and Synaptic distances of the Right Side of the nervous system were compared.

Description: Again, a positive correlation between developmental (lineage) distance and synaptic distance for pairs of neurons is sought, except in this case where only the right side of the nervous system is considered.

Method:

1. First, a file containing just the lineage strings corresponding to the right side of the nervous system was generated. A list of the neurons corresponding to the right side (Data file N.), composed of 96 neurons (typically neurons ending in 'R' for right side, such as 'AVER' and 'SAADR'), was fed into a program into which was also input a list of all neurons and their lineage strings (Data file N. "lineage280.txt"). Running the program "./newlineage RightNeurons.txt lineage280.txt RightLineage" under UNIX produced a text file 'RightLineage' which contained only the 96 lineage strings of the left side of the developing nervous system.
2. Next, the right lineage strings were loaded into a frame on Biobike. This was accomplished with the command: "(load-lineage-string :file "/home/ericw/c/worm/lindist/RightLineage")." This created a frameR slot with 96 right

neurons and their corresponding lineage strings.

3. Next, the lineage distance matrix was calculated for the right side of the nervous system with the command "(lindist FrameR)," where the resulting 96 x 96 matrix was stored in FrameR.
4. Next, the synaptic graph corresponding to the right side of the nervous system had to be generated. Since this was a subset of the graph of the entire nervous system, a program was written to take the list of right neurons, and the list of edges (synapses) of the entire nervous system, and generate from this the list of edges (synapses) that only refer to right neural nodes. This was accomplished with the command: "./newedges RightNeurons.txt AllEdges.txt RightEdges," under UNIX, where 'RightNeurons.txt' refers to Data File N (of the Appendix), and 'AllEdges.txt' refers to the set of synapses of the entire nervous system (Data File N of the Appendix), and 'RightEdges' is a text file which contains just the 840 synapses of the left side of the nervous system.
5. Next, the 96 neurons (nodes) and 931 synapses (edges) of the right side of the nervous system were loaded into a BioBike frame FrameS.
6. Next, the synaptic distance matrix was calculated for the left side of the nervous system. The command "(syndist :File "/home/ericw/c/worm/syndist/RightEdges")" was run from Biobike, generating a 96 x 96 matrix of synaptic (graph) distances, comparing each right neuron to every other right neuron.
7. Finally, a correlation between the lineage distance and synaptic distance of the right side of the nervous system was performed with the BioBike command "(fcor FrameR FrameS)."

Results:

Right Side Correlation of Right Lineage Distance Matrix with Right Synaptic Distance Matrix:

> <23>> (fcor #$Temp15 #$Temp17)
> :: (:R 0.05118071 :R2 0.0026194649)

(7) Synaptic-lineage correlation of the right side of the organism.

Discussion: Again the correlation between the right lineage distance matrix and right synaptic distance matrix is approximately zero, indicating no strong negative or positive correlation between these quantities.

Experiment Four: Lineage distances of the Right Side of the nervous system was compared to the Left Side of the nervous system.

Description: Developmental similarities between the left and right sides of the nervous system are sought. Compared are the lineage distances of for example ADAL to SAADL compared to ADAR to SAADR, i.e. distances between corresponding pairs of neurons on the left and right side of the nervous system.

Method:

1. First, two corresponding sets of neurons were created, one for the left and one for the right (named 'LeftCorrNeurons.txt' and 'RightCorrNeurons.txt,' respectively). By 'corresponding' is meant that the two lists consisted of the same number of neurons, in the same order, with corresponding neurons at the same position in the list, e.g. ADAL

in the first position in the left list, and ADAR in the first position in the right list, AIYL in the tenth position in the left list and AIYR in the tenth position in the right list, … URYVL in the last position in the left list and URYVR in the last position in the right list. In this way the same neurons with the same function in the left and right portions of the nervous system are being compared.

2. Next, the lineage list for the left side of the nervous system was generated with the command: "./newlineage LeftCorrNeurons.txt lineage280.txt LeftCorrLineage." 'LeftCorrNeurons.txt' refers to the left corresponding neuron list described above, 'lineage280.txt' refers to the total lineage file, and 'LeftCorrLineage' refers to the resulting text file of lineage strings with neurons that are part of the left corresponding neuron list only.
3. Next, the lineage list for the right side of the nervous system was generated with the command: "./newlineage RightCorrNeurons.txt lineage280.txt RightCorrLineage." 'RightCorrNeurons.txt' refers to the right corresponding neuron list described above, 'Lineage280.txt' refers to the total lineage string file, and 'RightCorrLineage' refers to the resulting text file of lineage strings with neurons that are part of the right corresponding neuron list only.
4. Next, the 'LeftCorrLineage' file was loaded into a BioBike frame, with the command: "load-lineage-string :file "/home/ericw/c/worm/lindist/LeftCorrLineage." This command loaded the left corresponding neuron list and corresponding lineage strings into a slot of FrameV.
5. Next, the 'RightCorrLineage' file was loaded into a BioBike frame, with the command:

"load-lineage-string :file "/home/ericw/c/worm/lindist/RightCorrLineage." This command loaded the right corresponding neuron list with corresponding lineage strings into a slot of FrameW.

6. Next, the lineage distance matrix was calculated for the left portion of the nervous system through the BioBike interface with the command: "(lindist FrameV)." The resulting distance matrix is stored in the matrix slot of FrameV.
7. Next, the lineage distance matrix was calculated for the right portion of the nervous system through the BioBike interface with the command: "(lindist FrameW)." The resulting distance matrix is stored in the matrix slot of FrameW.
8. Finally, the BioBike application 'fcor' was run to compare or correlate the distance matrices for the left and right synaptic networks. This was done with the command: "(fcor FrameV FrameW)" in BioBike.

Results:

<6>> (fcor #$Temp3 #$Temp4)
:: (:R 0.7823205 :R2 0.6120254)

(8) Lineage distance correlation of the two sides of the organism.

Discussion: There appears to be a significant correspondence (positive correlation equals .78) between the left and right sides of the nervous system in terms of their development. This was to be expected as nervous systems are typically highly symmetrical both in terms of neuron layout and symmetrical development. This particular result indicates the symmetrical nature of the development of the left and right sides of the nervous system.

Experiment Five: Synaptic distances of the Right Side of the nervous system was compared to the synaptic distances to the Left Side of the nervous system.

Description: By comparing synaptic distance matrices of the left and right side of the nervous system, a form of graph is produced, and hence a synaptic network comparison. The number that corresponds to the distance between AVEL and SAADL is compared to the distance between AVER and SAADR, for example, i.e., corresponding pairs are compared. As expected these distances are similar even though there are differences in the graphs (e.g. there are 826 synapses on the left and 867 synapses on the right even between corresponding neuron sets). This is an indication that the left and right synaptic networks are roughly comparable, i.e., have roughly similar form.

Method:

1. First, two corresponding sets of neurons were created, one for the left and one for the right (named 'LeftCorrNeurons.txt' and 'RightCorrNeurons.txt,' respectively). By 'corresponding' is meant that the two lists consisted of the same number of neurons, in the same order, with corresponding neurons at the same position in the list, e.g. ADAL in the first position in the left list, and ADAR in the first position in the right list, AIYL in the tenth position in the left list and AIYR in the tenth position in the right list, … URYVL in the last position in the left list and URYVR in the last position in the right list. In this way the same neurons with the same function in the left and right portions of the nervous system are being compared.
2. Next, the synaptic network for the left list was generated with the command: "./newedges

LeftCorrNeurons.txt Edges.txt LeftCorrEdges." 'LeftCorrNeurons.txt' refers to the left corresponding neuron list described above, 'Edges.txt' refers to the total synaptic network file, and 'LeftCorrEdges' refers to the resulting text file of synapses with neurons that are part of the left corresponding neuron list only.

3. Next, the synaptic network for the right list was generated with the command: "./newedges RightCorrNeurons.txt Edges.txt RightCorrEdges." 'RightCorrNeurons.txt' refers to the left corresponding neuron list described above, 'Edges.txt' refers to the total synaptic network file, and 'RightCorrEdges' refers to the resulting text file of synapses with neurons that are part of the right corresponding neuron list only.
4. Next, the 'LeftCorrEdges' file was loaded into a BioBike frame, with the command: "load-test-worm :file "/home/ericw/c/worm/syndist/LeftCorrEdges." This command loaded the left corresponding neuron list into the node slot and the list of synapses (edges) into the edge slot of FrameT.
5. Next, the 'RightCorrEdges' file was loaded into a BioBike frame, with the command: "load-test-worm :file "/home/ericw/c/worm/syndist/RightCorrEdges." This command loaded the right corresponding neuron list into the node slot and the list of synapses (edges) into the edge slot of FrameU.
6. Next, the distance matrix was calculated for the left portion of the nervous system through the BioBike interface with the command: "(syndist FrameT)." The resulting distance matrix is stored in the matrix slot of FrameT.
7. Next, the distance matrix was calculated for the right portion of the nervous system through the BioBike interface with the command: "(syndist FrameU)." The resulting

distance matrix is stored in the matrix slot of FrameU.

8. Finally, the BioBike application 'fcor' was run to compare or correlate the distance matrices for the left and right synaptic networks. This was done with the command: "(fcor FrameT FrameU)" in BioBike.

Results:

<11>> (fcor #$Temp1 #$Temp2)
:: (:R 0.6625694 :R2 0.43899822)

(9) Synaptic distance correlation of both sides of the organism.

Discussion: This time there is a positive correlation of .66 between the synaptic distances of the left side of the nervous system and the right side, or corresponding sets of neurons. Again this result is to be expected as nervous systems are in general highly symmetrical in terms of their synaptic development. The number .66 is not however exceptionally high as it is also true of nervous systems that although the nervous system is generally symmetrical, it is not entirely so. Examples of this are seen in the difference of number of incoming and outgoing connections to the central switch AVAL, which is 549, and AVAR, its right side counterpart, which has only 294 connections. Other examples include AIBL, with 319 connections, and AIBR with only 178 connections; PVCL with 187 connections, and PVCR with only 52 connections; and HSNL with 74 connections, and HSNR, with 120 connections. I.e., the nervous system demonstrates a gross symmetry, but the wiring is more complex than simple symmetry, and involves a fairly large number of exceptions, resulting in a positive correlation of just .66.

Controls

Two control experiments were run, comparing lineage and synaptic distances on opposite sides of the nervous system. The results were as follows from the left lineage and right synaptic distance matrices:

<9>> (fcor #$Temp2 #$Temp3)
:: (:R 0.052959997 :R2 0.0028047613)

(10) Correlation control.

Result: The result was again a near zero correlation between lineage and synaptic distance. The result from the right lineage and left synaptic distance matrices:

<10>> (fcor #$Temp1 #$Temp4)
:: (:R 0.058016494 :R2 0.0033659136)

(11) Correlation control.

Result: The result was again a near zero correlation between lineage and synaptic distances of opposite sides of the nervous system.

Summary

In summary, no significant correlation between lineage distance and synaptic distance was found, even when these two quantities were correlated for just one side of the nervous system. It is to be remembered that correlation merely looks for linear relationships in the data, and no such linear relationship was found. It is entirely possible that a curvilinear relationship holds between these data sets, however searching for such a curvilinear relationship with a standard plotting tool (PSI-Plot) also yielded no results. Some work was done with self-organizing maps to look for more complex relationships between the lineage and synaptic distance data sets, and preliminary results indicated that some relationship, still to be discerned,

was present. There are biological reasons why the relationship between synaptic distance and lineage distance is not simple if present at all. Experimental data indicate that many neurons migrate through the nervous system before establishing synaptic contacts making lineage non-position specific. This is true, for example of HSNL, which migrates to the vulval area before sending out projections to innervate motor neurons in this area. Secondly, axonal projections may be lengthy and the neurons upon which a neuron synapses may be distant from the cell's development locus. On the other hand, the correlations in this study did indicate significant developmental and synaptic symmetry in the *C. elegans* nervous system. Lineage distances on the right side of the nervous system showed a positive correlation with lineage distances on the left side of the nervous system. Likewise, synaptic distances on the right side of the nervous system show a more modest positive correlation with synaptic distances on the left side of the nervous system, indicating quasi-symmetrical synaptic development.

SUMMARY

It is important to understand that in this study a set of computational tools was developed that not only enabled the analyses performed in this study, but which also are capable of other future analyses. The tools may be used to analyze specific neural circuits in *C. elegans*, to analyze different development stages of the organism, and can potentially analyze nervous systems of other types of animals. Specifically analyzed in this study was the development and synaptic network of the entire organism.

In the developmental model, the output of the computer program indicates which neurons are born at which stages, and which genes are activated in these neurons. Also given as output of the program is the synaptic network for each development stage. Although this information is just touched upon in the text, it is important novel information for the developmental study of the *C. elegans* nervous system. The stage specific synaptic networks are predicted synaptic networks, and due to cell migration may differ somewhat form the actual stage-specific synaptic network. The stage specific synaptic networks are given on the CD as the graph file which describes them is to big to be presented in this paper.

The synaptic pathway analysis introduces nine novel programs that may be used to investigate and characterize stage-specific synaptic networks. In this analysis, the mature synaptic network of the adult was always used. These tools indicated which pathways exist between input sensory neurons and output motor neurons. Analyses on specific circuits may be performed. These tools also indicated the genetic makeup of the pathways. A tool was provided that identifies high throughput neurons and subpaths and ganglia. Another tool was provided which identified feedback mechanisms in the nervous system.

Finally an attempt was made to look for relationships between development (lineage strings) and synaptic development (synaptic network graphs). No such relationships were clearly discerned, but the correlations only looked for linear relationships.

This project will continue in the form of the Biobike computing platform. The programs here written will be incorporated into the Biobike web interface for other biologists to use in their investigations of the *C. elegans* nervous system.

Appendix I. *C. elegans* Neuron Descriptions

The C elegans Parts List
From Sulston, JE and White, JG (1988), "Parts list", in The Nematode Caenorhabditis elegans, eds WB Wood et al, Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York, USA, pp 415 - 431.

Cell	Description
ADAL	Ring interneurons
ADAR	Ring interneurons
ADEL	Anterior deirid, sensory neuron, dopaminergic
ADER	Anterior deirid, sensory neuron, dopaminergic
ADFL	Amphid neuron, prob. chemosensory
ADFR	Amphid neuron, prob. chemosensory
ADLL	Amphid neuron, prob. chemosensory
ADLR	Amphid neuron, prob. chemosensory
AFDL	Amphid finger cell, neuron associated with amphid sheath
AFDR	Amphid finger cell, neuron associated with amphid sheath
AIAL	Amphid interneuron
AIAR	Amphid interneuron
AIBL	Amphid interneuron
AIBR	Amphid interneuron
AIML	Ring interneuron
AIMR	Ring interneuron
AINL	Ring interneuron
AINR	Ring interneuron
AIYL	Amphid interneuron
AIYR	Amphid interneuron
AIZL	Amphid interneuron
AIZR	Amphid interneuron
ALA	Neuron, sends processes laterally and along dorsal cord
ALML	Anterior lateral microtubule cell, touch receptor
ALMR	Anterior lateral microtubule cell, touch receptor
ALNL	Neuron associated with ALM
ALNR	Neuron associated with ALM
AQR	Neuron, basal body. not part of a sensillum, projects into ring
AS1	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS2	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS3	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS4	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS5	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS6	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS7	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS8	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS9	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS10	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
AS11	Ventral cord motor neuron, innervates dorsal muscles, no ventral counterpart
ASEL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASER	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil

ASGL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASGR	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASHL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASHR	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASIL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASIR	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASJL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASJR	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASKL	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
ASKR	Amphid neurons, single ciliated endngs, probably chemo-sensory; project into ring via commissure from ventral ganglion, make diverse synaptie connections in ring neuropil
AUAL	Neuron, process runs with amphid processes but lacks cillialted ending
AUAR	Neuron, process runs with amphid processes but lacks cillialted ending
AVAL	Ventral cord interneuron, synapses onto VA, DA, and AS motor neurons; formerly called alpha
AVAR	Ventral cord interneuron, synapses onto VA, DA, and AS motor neurons; formerly called alpha
AVBL	Ventral cord interneuron, synapsesonto VB and DB motor neurons; formerly called beta
AVBR	Ventral cord interneuron, synapsesonto VB and DB motor neurons; formerly called beta
AVDL	Ventral cord interneuron, synapses onto VA, DA, AS motorneurons; formerly called delta
AVDR	Ventral cord interneuron, synapses onto VA, DA, AS motorneurons; formerly called delta
AVEL	Ventral cord interneuron, like AVD but outputs restrict-ed to anterior cord
AVER	Ventral cord interneuron, like AVD but outputs restrict-ed to anterior cord
AVFL/R	Interneuron, processes in ventral cord and ring, few synapses
AVFL/R	Interneuron, processes in ventral cord and ring, few synapses
AVG	Ventral cord interneuron, few synapses
AVHL	Neuron, mainly postsynaptic in ventral cord and presynaptic in the ring
AVHR	Neuron, mainly postsynaptic in ventral cord and presynaptic in the ring
AVJL	Neuron, synapses like AVHL/R
AVJR	Neuron, synapses like AVHL/R
AVKL	Ring and ventral cord interneuron
AVKR	Ring and ventral cord interneuron
AVL	Ring and ventral cord interneuron, few synapses
AVM	Anterior ventral microtubule cell, touch receptor
AWAL	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with amphid sheath
AWAR	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with amphid sheath
AWBL	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with amphid sheath
AWBR	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with amphid sheath
AWCL	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with amphid sheath
AWCR	Amphid wing cells, neurons having ciliated sheet-like sensory endings closely associated with

	amphid sheath
BAGL	Neuron, ciliated ending in head, no supporting cells, associated with ILso
BAGR	Neuron, ciliated ending in head, no supporting cells, associated with ILso
BDUL	Neuron, process runs along excretory canal and into ring, unique darkly staining synaptic vesicles
BDUR	Neuron, process runs along excretory canal and into ring, unique darkly staining synaptic vesicles
CANL	Process runs along excretory canal, no synapses, essential for survival
CANR	Process runs along excretory canal, no synapses, essential for survival
CEPDL	Cephalic neurons, contain dopamine
CEPDR	Cephalic neurons, contain dopamine
CEPVL	Cephalic neurons, contain dopamine
CEPVR	Cephalic neurons, contain dopamine
DA1	Ventral cord motor neurons, innervate dorsal muscles
DA2	Ventral cord motor neurons, innervate dorsal muscles
DA3	Ventral cord motor neurons, innervate dorsal muscles
DA4	Ventral cord motor neurons, innervate dorsal muscles
DA5	Ventral cord motor neurons, innervate dorsal muscles
DA6	Ventral cord motor neurons, innervate dorsal muscles
DA7	Ventral cord motor neurons, innervate dorsal muscles
DA8	Ventral cord motor neurons, innervate dorsal muscles
DA9	Ventral cord motor neurons, innervate dorsal muscles
DB1/3	Ventral cord motor neurons, innervate dorsal muscles
DB2	Ventral cord motor neurons, innervate dorsal muscles
DB3/1	Ventral cord motor neurons, innervate dorsal muscles
DB4	Ventral cord motor neurons, innervate dorsal muscles
DB5	Ventral cord motor neurons, innervate dorsal muscles
DB6	Ventral cord motor neurons, innervate dorsal muscles
DB7	Ventral cord motor neurons, innervate dorsal muscles
DD1	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DD2	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DD3	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DD4	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DD5	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DD6	Ventral cord motor neurons, reciprocal inhibitors, change synaptic pattern during postembryonic development
DVA	Ring interneurons, cell bodies in dorsorectal ganglion
DVB	Ring interneuron, cell body in dorsorectal ganglion, inervates rectal muscles
DVC	Ring interneurons, cell bodies in dorsorectal ganglion
FLPL	Neuron, ciliated ending in head, no supporting cells, associated with ILso
FLPR	Neuron, ciliated ending in head, no supporting cells, associated with ILso
HSNL	Herm. specific motor neurons (die in male embryo), innervate vulval muscles, serotonergic
HSNR	Herm. specific motor neurons (die in male embryo), innervate vulval muscles, serotonergic
IL1DL	Inner labial neuron
IL1DR	Inner labial neuron
IL1L	Inner labial neuron
IL1R	Inner labial neuron
IL1VL	Inner labial neuron
IL1VR	Inner labial neuron

IL2DL	Inner labial neuron
IL2DR	Inner labial neuron
IL2L	Inner labial neuron
IL2R	Inner labial neuron
IL2VL	Inner labial neuron
IL2VR	Inner labial neuron
LUAL	Interneuron, short process in post ventral cord
LUAR	Interneuron, short process in post ventral cord
OLLL	Lateral outer labial neurons
OLLR	Lateral outer labial neurons
OLQDL	Quadrant outer labial neuron
OLQDR	Quadrant outer labial neuron
OLQVL	Quadrant outer labial neuron
OLQVR	Quadrant outer labial neuron
PDA	Motor neuron, process in dorsal cord, same as Y cell in hermaphrodite, Y.a in male
PDB	Motor neuron, process in dorsal cord, cell body in pre-anal ganglion
PDEL	Neuron, dopaminergic of postderid sensillum
PDER	Neuron, dopaminergic of postderid sensillum
PHAL	Phasmid neurons, probably chemosensory
PHAR	Phasmid neurons, probably chemosensory
PHBL	Phasmid neurons, probably chemosensory
PHRB	Phasmid neurons, probably chemosensory
PHCL	Neuron, striated rootlet in male, possibly sensory in tail spike
PHCR	Neuron, striated rootlet in male, possibly sensory in tail spike
PLML	Posterior lateral microtubule cell, touch receptor
PLMR	Posterior lateral microtubule cell, touch receptor
PLNL	Interneuron, associated with PLM
PLNR	Interneuron, associated with PLM
PQR	Neuron, basal body, not part of a sensillum, projects into preanal gangion
PVCL	Ventral cord interneuron, cell body in lumbar ganglion, synapses onto VB andDB motor neurons, formerly called delta.
PVCR	Ventral cord interneuron, cell body in lumbar ganglion, synapses onto VB andDB motor neurons, formerly called delta.
PVDL	Neuron, lateral process adjacent to excretory canal
PVDR	Neuron, lateral process adjacent to excretory canal
PVM	Posterior ventral microtuble cell, touch receptor
PVNL	Interneuron/motor neuron, post. vent. cord, few synapses
PVNR	Interneuron/motor neuron, post. vent. cord, few synapses
PVPL	Interneuron, cell body in lumbar ganglion, projects along ventral cord to nerve ring
PVPR	Interneuron, cell body in lumbar ganglion, projects along ventral cord to nerve ring
PVQL	Interneuron, projects along ventral cord to ring
PVQR	Interneuron, projects along ventral cord to ring
PVR	Interneuron, projects along ventral cord to ring
PVT	Interneuron, projects along ventral cord to ring
PVWL	Interneuron, posterior ventral cord, few synapses
PVWR	Interneuron, posterior ventral cord, few synapses
RIAL	Ring interneuron, many synapses
RIAR	Ring interneuron, many synapses
RIBL	Ring interneuron
RIBR	Ring interneuron
RICL	Ring interneuron
RICR	Ring interneuron

RID	Ring interneuron, projects along dorsal cord
RIFL	Ring interneuron
RIFR	Ring interneuron
RIGL	Ring interneuron
RIGR	Ring interneuron
RIH	Ring interneuron
RIML	Ring motor neuron
RIMR	Ring motor neuron
RIPL	Ring/pharynx interneuron, only direct connection between pharynx and ring
RIPR	Ring/pharynx interneuron, only direct connection between pharynx and ring
RIR	Ring interneuron
RIS	Ring interneuron
RIVL	Ring interneuron
RIVR	Ring interneuron
RMDDL	Ring motor neuron/interneuron, many synapses
RMDDR	Ring motor neuron/interneuron, many synapses
RMDL	Ring motor neuron/interneuron, many synapses
RMDR	Ring motor neuron/interneuron, many synapses
RMDVL	Ring motor neuron/interneuron, many synapses
RMDVR	Ring motor neuron/interneuron, many synapses
RMED	Ring motor neuron
RMEL	Ring motor neuron
RMER	Ring motor neuron
RMEV	Ring motor neuron
RMFL	Ring motor neuron/interneuron
RMFR	Ring motor neuron/interneuron
RMGL	Ring interneuron
RMGR	Ring interneuron
RMHL	Ring motor neuron/interneuron
RMHR	Ring motor neuron/interneuron
SAADL	Ring interneuron, anteriorly projecting process that runs sublaterally
SAADR	Ring interneuron, anteriorly projecting process that runs sublaterally
SAAVL	Ring interneuron, anteriorly projecting process that runs sublaterally
SAAVR	Ring interneuron, anteriorly projecting process that runs sublaterally
SABD	Ring interneuron, anteriorly projecting process that runs sublaterally, synapses to anterior body muscles in L1
SABVL	Ring interneuron, anteriorly projecting process that runs sublaterally, synapses to anterior body muscles in L1
SABVR	Ring interneuron, anteriorly projecting process that runs sublaterally, synapses to anterior body muscles in L1
SDQL	Post. lateral interneuron, process projects into ring
SDQR	Ant. lateral interneuron, process projects into ring
SIADL	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIADR	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIAVL	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIAVR	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIBDL	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIBDR	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIBVL	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SIBVR	Receives a few synapses in the ring, has a posteriorly directed process that runs sublaterally
SMBDL	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMBDR	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally

SMBVL	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMBVR	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMDDL	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMDDR	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMDVL	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
SMDVR	Ring motor neuron/interneuron, has a posteriorly direct ed process that runs sublaterally
URADL	Ring motor neuron
URADR	Ring motor neuron
URAVL	Ring motor neuron
URAVR	Ring motor neuron
URBL	Neuron, presynaptic in ring, ending in head
URBR	Neuron, presynaptic in ring, ending in head
URXL	Ring interneuron
URXR	Ring interneuron
URYDL	Neuron, presynaptic in ring, ending in head
URYDR	Neuron, presynaptic in ring, ending in head
URYVL	Neuron, presynaptic in ring, ending in head
URYVR	Neuron, presynaptic in ring, ending in head
VA1	Vent. cord motor neuron, innervates vent. body muscles
VA2	Vent. cord motor neuron, innervates vent. body muscles
VA3	Vent. cord motor neuron, innervates vent. body muscles
VA4	Vent. cord motor neuron, innervates vent. body muscles
VA5	Vent. cord motor neuron, innervates vent. body muscles
VA6	Vent. cord motor neuron, innervates vent. body muscles
VA7	Vent. cord motor neuron, innervates vent. body muscles
VA8	Vent. cord motor neuron, innervates vent. body muscles
VA9	Vent. cord motor neuron, innervates vent. body muscles
VA10	Vent. cord motor neuron, innervates vent. body muscles
VA11	Vent. cord motor neuron, innervates vent. body muscles
VA12	Vent. cord motor neuron, innervates vent. body muscles, but also interneuron in preanal ganglion
VB1	Vent. cord motor neuron, innervates vent. body muscles, also interneuron in ring
VB2	Vent. cord motor neuron, innervates vent. body muscles
VB3	Vent. cord motor neuron, innervates vent. body muscles
VB4	Vent. cord motor neuron, innervates vent. body muscles
VB5	Vent. cord motor neuron, innervates vent. body muscles
VB6	Vent. cord motor neuron, innervates vent. body muscles
VB7	Vent. cord motor neuron, innervates vent. body muscles
VB8	Vent. cord motor neuron, innervates vent. body muscles
VB9	Vent. cord motor neuron, innervates vent. body muscles
VB10	Vent. cord motor neuron, innervates vent. body muscles
VB11	Vent. cord motor neuron, innervates vent. body muscles
VC1	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VC2	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VC3	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VC4	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VC5	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VC6	Hermaphrodite specific vent cord motor neuron innervates vulval muscles and vent body muscles
VD1	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD2	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD3	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD4	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD5	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor

VD6	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD7	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD8	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD9	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD10	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD11	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD12	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor
VD13	Vent cord motor neuron, innervates vent body muscles, probably reciprocal inhibitor

Appendix II. Developmental Model Results

Table 10. Stage One Neurons

All Sensory neurons at this stage:
ADFL ADFR ADLL ADLR AFDL AFDR ALML ALMR ASGL ASGR ASHL ASHR AWAL AWAR AWBL AWBR AWCL AWCR PHAL PHAR
All Motor neurons at this stage:
DA1 DA2 DA3 DA4 DA5 DA6 DA7 DA8 DA9 DB1 DB2 DB3 DB4 DB5 DB6 DB7 DD1 DD2 DD3 DD4 DD5 DD6 PDA RIML RIMR RMDDL RMDDR RMDL RMDR RMDVL RMDVR RMED RMEL RMER RMEV URADL URADR URAVL URAVR
All Inter-neurons at this stage:
AIAL AIAR AIBL AIBR AIML AIMR AINL AINR AIYL AIYR ALA AVAL AVAR AVBL AVBR AVDL AVDR AVEL AVER AVG AVHL AVHR AVJL AVJR AVKL AVKR AVL BAGL BAGR BDUL BDUR DVA DVB IL2DL IL2DR IL2L IL2R IL2VL IL2VR PVPL PVPR PVQL PVQR PVT RIAL RIAR RIBL RIBR RID RIGL RIGR RIH RIPL RIPR RIR RIS RIVL RIVR RMGL RMGR SAADL SAADR SAAVL SAAVR SABD SIADL SIADR SIAVL SIAVR SIBDL SIBDR SIBVL SIBVR SMBVL SMBVR SMDDL SMDDR SMDVL SMDVR URBL URBR URYDL URYDR URYVL URYVR

Table 11. Sensory-specific genes of Stage One.

Sensory-specific genes:
bra-1 ceh-32 ceh-36 ceh-37 che-1 che-2 che-3 daf-11 daf-5 dyn-1 egl-43 egl-46 gcy-12 gpa-1 gpa-10 gpa-11 gpa-13 gpa-15 gpa-5 gpa-6 gpc-1 grk-2 lst-1 mec-10 mec-2 mec-3 mec-4 mec-6 mec-8 mig-2 mps-1 mtd-1 nhr-79 nid-1 nlp-10 nlp-14 nlp-7 nlp-8 nlp-9 nsy-1 ocr-1 ocr-2 odr-1 odr-10 odr-3 odr-7 osm-10 osm-9 pat-4 ptl-1 sgk-1 sra-13 srb-6 srd-1 sre-1 srg-13 srh-220 sro-1 srp-2 str-1 str-2 tag-97 tba-1 tbx-2 unc-32 unc-73 unc-97 ver-2

Table N. Motor genes in stage one.

Motor-specific genes
ace-3 clh-6 daf-9 itr-1 mig-13 rig-5 smp-2 unc-129 unc-4 vab-7
Motor Neuron Genes at this stage
ace-2 ace-3 ahr-1 avr-15 cam-1 cav-1 cdc-42 ced-10 ceh-10 ceh-17 ceh-6 cfi-1 cha-1 chd-3 clh-6 csk-1 daf-9 dbl-1 dop-1 dop-2 egl-4 egl-5 fbxb-103 flp-18 flp-21 glr-1 glr-2 glr-4 glr-5 goa-1 ina-1 itr-1 kvs-1 lim-4 lim-6 lin-35 lin-53 mgl-1 mgl-2 mig-13 nmr-1 nmr-2 npr-1 odr-2 ptp-3 qui-1 rig-5 sax-3 ser-2 set-2 slt-1 smp-2 src-1 tax-6 unc-129 unc-17 unc-25 unc-4 unc-40 unc-42 unc-47 unc-5 unc-53 unc-8 unc-86 vab-7

Table N. Interneuron-specific genes at stage one.

Inter-neuron specific genes:
ceh-24 ceh-43 cnb-1 daf-14 deg-1 eat-20 egl-36 fax-1 flp-1 gcy-33 gcy-35 ggr-1 ggr-2 glr-3 glr-6 gly-18 gpa-16 gpa-9 hen-1 hid-1 hlh-14 igcm-2 kal-1 lad-2 let-60 nhr-22 nlp-11 nlp-6 oig-1 oig-3 opt-3 rig-3 ser-4

smp-1 sng-1 snt-1 sra-10 sra-11 sup-9 tba-2 trp-1 unc-6 unc-71 unc-93 vab-8 ver-3 wrk-1 zig-1 zig-2 zig-3 zig-4 zig-5 zig-8

Table N. Genes Common to Stage One

Genes common to sensory neurons, motor neurons, and interneurons at this stage:
ace-2 cam-1 cav-1 cdc-42 ced-10 chd-3 csk-1 dop-1 egl-4 fbxb-103 goa-1 ina-1 lim-4 lim-6 lin-35 lin-53 npr-1 odr-2 ptp-3 set-2 src-1 tax-6 unc-40 unc-42 unc-8 unc-86

Table N. Sensory neurons at stage two.

New sensory neurons at this stage:
ADEL ADER ASEL ASER ASIL ASIR ASJL ASJR ASKL ASKR PHBL PHBR

Table N. New sensory specific genes in stage two.

New sensory-specific genes:
bra-1 ceh-14 ceh-23 ceh-36 daf-11 daf-28 daf-4 daf-7 egl-4 flp-21 gcs-1 gcy-5 gcy-6 gcy-7 gpa-1 gpa-13 gpa-14 gpa-15 gpa-2 gpa-4 gpa-5 gpa-6 gpa-9 gpc-1 hen-1 kvs-1 lim-6 lsy-6 mps-1 ncs-1 nlp-1 nlp-10 nlp-14 nlp-18 nlp-24 nlp-27 nlp-5 nlp-6 nlp-7 nlp-8 nlp-9 ocr-2 odr-1 osm-10 osm-3 pkc-1 skn-1 sra-6 sra-7 sra-9 srb-6 srd-1 sre-1 srg-2 srg-8 srp-2 str-2 str-3 tax-2 tbx-2 ttx-3 unc-3 unc-37 wrk-1 zig-3 zig-4 zig-5

Newly activated sensory genes this stage:
daf-28 daf-7 gcs-1 gcy-5 gcy-6 gcy-7 gpa-4 lsy-6 nlp-24 nlp-27 nlp-5 skn-1 sra-7 sra-9 srg-8 str-3 unc-3 unc-37

Table N. Motor neurons born in stage two.

New motor neurons at this stage:
HSNL HSNR OLLL OLLR OLQDL OLQDR OLQVL OLQVR RMFL RMFR RMHL RMHR VA1 VD1

Table N. New motor and newly activated motor genes in stage two.

New motor-specific genes:
bbs-5 cat-1 cdh-3 cha-1 che-13 clh-3 eat-16 eat-20 egl-21 egl-43 flt-1 gar-2 ggr-2 gpb-2 grd-6 gsa-1 ham-2 hbl-1 ins-11 ins-2 ins-22 ins-23 ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 inx-4 jkk-1 jnk-1 mab-23 mig-1 mig-2 nhx-5 nid-1 nlp-15 ocr-4 osm-5 rgs-2 sem-4 syg-1 tph-1 unc-14 unc-17 unc-2 unc-25 unc-4 unc-47 unc-51 unc-53 unc-73 unc-76
Newly activated motor genes this stage:
bbs-5 cdh-3 che-13 clh-3 eat-16 flt-1 gar-2 gpb-2 grd-6 gsa-1 ham-2 hbl-1 ins-11 ins-2 ins-22 ins-23 ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 inx-4 mab-23 mig-1 nhx-5 ocr-4 osm-5 rgs-2 sem-4 syg-1 unc-14 unc-51 unc-76

Table N. Interneurons born in stage two.

New inter-neurons at this stage:
ADAL ADAR AIZL AIZR AUAL AUAR CEPDL CEPDR CEPVL CEPVR FLPL FLPR IL1DL IL1DR IL1L IL1R IL1VL IL1VR LUAL LUAR PVCL PVCR RICL RICR RIFL RIFR SABVL SABVR SMBDL SMBDR URXL URXR

Table N. Active and newly activated interneuron genes in stage two.

New interneuron-specific genes:

ace-2 ceh-10 ceh-6 cfi-1 cnb-1 daf-5 daf-9 deg-1 deg-3 des-2 dop-1 fax-1 flr-4 gcy-32 gcy-34 gcy-35 gcy-36 gcy-37 glr-1 glr-2 glr-4 gpa-16 gpa-8 hlh-2 itr-1 let-60 lin-11 mab-21 mab-9 mdl-1 mec-10 mec-3 mec-7 nhr-83 nlp-13 nmr-1 nmr-2 odr-2 rig-1 rig-6 sng-1 sra-10 sre-37 unc-42 unc-6
Newly activated interneuron genes this stage:
des-2 flr-4 gcy-32 gcy-34 gcy-36 gcy-37 gpa-8 mab-21 mab-9 mdl-1 rig-1 rig-6 sre-37

Table N. Common genes of stage two.

Genes common to new neurons at this stage:
cam-1 cav-1 cdc-42 ced-10 chd-3 che-2 eat-4 ida-1 ina-1 kal-1 lin-35 lin-53 npr-1 osm-6 osm-9 set-2 unc-40

Table N. New sensory neurons and sensory neuron genes at stage three.

New sensory neurons at this stage:
PLML PLMR
New sensory-specific genes:
daf-1 deg-3 eat-4 egl-46 egl-5 glr-8 gpa-16 mec-2 mec-3 mec-4 mec-6 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1 ptl-1 rhgf-1 sax-7 tba-1 tba-2 unc-73 unc-97

Table N. Interneurons born at stage three.

New inter-neurons at this stage:
ALNL ALNR AQR PQR PVWL PVWR

Table N. Interneuron-specific genes active in stage three.

New interneuron-specific genes:
ace-3 ceh-13 ceh-14 cha-1 che-2 che-3 cki-1 daf-28 eat-20 egl-2 egl-36 egl-4 gcy-32 gcy-34 gcy-36 gcy-37 gpa-10 gpa-8 kal-1 lad-2 npr-1 odr-2 osm-6 ser-2 tax-2 unc-103 unc-17 unc-53

Table N. Sensory neurons born in stage four.

New sensory neurons at this stage:
AVM PVM

Table N. Sensory genes active in stage four.

New sensory-specific genes:
dop-1 eat-4 egl-46 flt-1 gar-1 goa-1 gpa-16 mab-21 mec-10 mec-2 mec-3 mec-4 mec-6 mec-7 mec-8 mtd-1 pat-4 ptl-1 sax-7 tba-1 tol-1 unc-40 unc-97

Table N. Motor neurons born in stage four.

New motor neurons at this stage:
AS1 AS2 AS3 AS4 AS5 AS6 AS7 AS8 AS9 AS10 AS11 PDB VB2 VC1 VC2 VC3 VC4 VC5 VC6 VD2 VD3 VD4 VD5 VD6 VD7 VD8 VD9 VD10 VD11 VD12 VD13

Table N. Motor-specific genes active in stage four.

New motor-specific genes:
cat-1 ced-10 dbl-1 del-1 exc-7 hmr-1 ida-1 lin-29 unc-25 unc-42 unc-47 vab-7

Table N. Inter-neurons born in stage four.

New inter-neurons at this stage:
AVFL AVFR PVNL PVNR SDQL SDQR

Table N. Inter-neuron-specific genes active in stage four.

New interneuron-specific genes:
ceh-14 cha-1 lad-2 ser-2 unc-17 unc-73

Table N. Common genes in stage four.

Genes common to new neurons at this stage:
pag-3

Table N. Sensory neurons born at stage five and their genes.

New sensory neurons at this stage:
PDEL PDER PHCL PHCR
New sensory-specific genes
cat-2 ceh-14 che-2 che-3 dat-1 dop-2 egl-36 gpa-16 ida-1 jkk-1 jnk-1 mab-23 osm-6 pkc-1 unc-40 unc-73

Table N. Motor neurons born at stage five and their genes.

New motor neurons at this stage:
VA2 VA3 VA4 VA5 VA6 VA7 VA8 VA9 VA10 VA11 VA12 VB1 VB3 VB4 VB5 VB6 VB7 VB8 VB9 VB10 VB11
New motor-specific genes:
avr-15 dbl-1 del-1 exc-7 lin-29 mab-21 pag-3 unc-42 unc-6

Table N. Inter-neurons born at stage five and their genes.

New inter-neurons at this stage:
PLNL PLNR
New interneuron-specific genes:
cha-1 daf-1 egl-2 gcy-35 lad-2 odr-2 unc-17

Table N. Inter-neurons born at stage six and their genes.

New inter-neurons at this stage:
DVC PVDL PVDR PVR
New interneuron-specific genes:
cam-1 cav-1 cdc-42 ced-10 ceh-14 chd-3 deg-3 des-2 eat-4 egl-4 egl-46 fbxb-103 ggr-1 glr-1 glr-4 goa-1 hcp-3 ina-1 jkk-1 jnk-1 kal-1 lin-11 lin-35 lin-53 mec-10 mec-12 mec-3 mec-6 mec-7 mec-9 nhr-83 nlp-11 osm-9 pkc-1 ptp-3 sem-4 ser-2 ser-4 set-2 unc-103 unc-32 unc-40 unc-86

Appendix III

Computer Programs

1. load-test-worm() [LISP]
2. load-lineage-string() [LISP]
3. lindist() [LISP]
4. LinDistCL.c [C]
5. syndist() [LISP]
6. SynDistCL.cpp() [C++]
   a. operator>> [C++]
      i. clear() [C++]
      ii. insertVertex [C++]
      iii.insertEdge [C++]
   b. shortestPath [C++]
      i. getvInfoIndex [C++]
   c. Graph.h [C++]
7. fcor() [LISP]
8. newlineage() [C++]
9. newedges() [C++]
10. strongcomponent() [C++]
    a. strongComponents() [C++]
       i. dfsVisit() [C++]
11. neuronpaths() [C++]
12. neuronenv() [C++]
13. neuronenvgenes() [C++]
14. subpaths() [C++]
15. pathgenes() [C++]
16. pathgenespairs [C++]
17. develstages() [C++]

1. LISP program: load-test-worm()

```
(defun load-test-worm (&key (file "/home/ericw/c/worm/syndist/Syn2.txt"))
  "Just for testing purposes, loads up the graph from Eric's input example in the format required by syndist"
  (with-open-file
   (i file)
   (let ((frame (make-temp-frame)))
     (setf (#^nodes frame)
           (loop as k below (parse-integer (read-line i nil nil))
                 for line = (read-line i nil nil)
                 collect (read-from-string line)))
     (setf (#^edges frame)
           (loop as k below (parse-integer (read-line i nil nil))
                 for line = (read-line i nil nil)
                 collect (let ((entry (mapcar #'read-from-string
                                              (string-split line #\tab))))
                           (list (first entry) (third entry) (second entry)))))
     frame)))
```

2. LISP Program: load-lineage-string()

```
(defun load-lineage-string
      (&key (file "/home/ericw/c/worm/lindist/NeuronGood.txt"))
      "Just for testing purposes, loads up NeuronGood.txt in the format required by
lindist"
      (with-open-file (i file)
        (let ((frame (make-temp-frame)))
          (setf (#^lineagestring frame)
                (loop as k below (parse-integer (read-line i nil nil))
                      for line = (read-line i nil nil) collect
                      (remove "" (string-split line #\Space)
                        :test #'string-equal)))
          frame)))
```

3. LISP Program: lindist()

This LISP program calls the C Program lindist() from the Biobike interface.

```
(defun lindist (frame)
      "Take a graph frame, as results, for example, from
(load-lineage-string), run
   it through lindist, and get the resulting matrix into the slot
#^matrix of
   the given frame."
      (with-temp-file-in (lindist-input
                     common-lisp-user:*tmp-directory* :type
                     "ldin")
        (with-temp-file-in (lindist-output
                      common-lisp-user:*tmp-directory* :type
                      "ldout")
          (with-temp-file-in (lindist-log
                       common-lisp-user:*tmp-directory* :type
                       "ldlog")
            (with-open-file (o lindist-input :direction :output
                        :if-exists :supersede)
              (format o "~a~%" (length (#^lineagestring frame)))
              (loop for (gene lineage) in (#^lineagestring frame) do
                   (format o "~a~c~a~%" gene #\Tab lineage)))
            (excl:run-shell-command
              (print (format nil
                       "cd /home/ericw/c/worm/lindist;./lindist ~a ~a > ~a"
                       lindist-input lindist-output
                       lindist-log)))
            (with-open-file (i lindist-output)
              (setf (#^matrix frame)
                   (loop for string in (#^lineagestring frame)
                        collect
                        (read-from-string
                          (format nil "(~a)"
                                (read-line i nil nil)))))))
          frame)))
```

4. C/UNIX Program LinDistCL.c:

```
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXNODES 300  // 25 max ok, 50 max runs into mem model probs
#define MAXCONNECTIONS MAXNODES
#define STRINGSIZE 25
#define TRUE 1
#define FALSE 0
#define YES 1
#define NO 0
#define DELETED 0
#define ACTIVE 1
#define LINESIZE 80
#define NUMTOKENS 36
#define INFINITY 999
#define MEMBER 1
#define NONMEMBER 0
#define MAXITERATIONS 10
void printmenu(void);
void deletenode(void);


// Stanford Work.  This version is modified to calculate lineage
// distance AND write out the lineage distance file.



      struct node {
            // whatever information, variables each node contains
            char cellname[STRINGSIZE];
                int active;  // marks node as active 1 or deleted 0
            char lineagestring[STRINGSIZE];
            float  timing;
            char descriptor[STRINGSIZE];
            };

      struct arc {
            // whatever information each connection between
            //   nodes contains, e.g. 'relations' information
               int connect; // specifies if 2 nodes are connected = 1
                            //     or not = 0, very important
               int distance;
               float timing;
               // for every nodes[i][j] != 0, there is a relation-
            //          ship type
               };


            // This is a C representation of a graph structure
```

```
      struct graph {
            struct node nodes[MAXNODES]; // declar # of nodes as array of
structs
            struct arc arcs[MAXNODES][MAXCONNECTIONS]; // There are
CONNECTIONS
                 // connections from each node to all other nodes
                };
      struct graph g;    //  an abbrev graph identifier
      struct node *nodeptr = &g.nodes[0];
      // Synaptic Connectivity Graph
      struct graph g1;
      struct graph g2;

      // Global variable declarations

      int i, j, NUMNODES;
      // const int CONNECTIONS=NUMNODES;
      int OLDNUMNODES = 0;
      int NUMRELATIONS = 0;
      int FILEREADLOOP;
      int yes = 0;
        char string1[STRINGSIZE];
        char string2[STRINGSIZE];

// FUNCTIONS:

int shorter_string(char* x, char* y)
{
      int a = 0, b = 0;
      a = strlen(x);
      b = strlen(y);
      if(a < b)
            return a;
      else if(b < a)
            return b;
      else
            return a;
}

int longer_string(char* x, char* y)
{
      int a = 0, b = 0;
      a = strlen(x);
      b = strlen(y);
      if(a > b)
            return a;
      else if(b > a)
            return b;
      else
            return a;
}
```

```
int lineage_distance( char* x, char* y)
{
      int length = longer_string(x, y);
      int i = 0, j = 0, k = 0, start=0;

      for(i = 0; i < length; ++i)
      {
            if(x[i] != y[i])
            {
                  for(j=i; j<length; ++j)
                        ++k;
                  break;
            }
      }
      return k;
}

FILE *ingraph;
FILE *outgraph;
char ingraphname[STRINGSIZE];
char outgraphname[STRINGSIZE];

int main(int argc,char *argv[])
{
// int argc;
// char *argv[3];
int i = 0;
int j = 0;


 if(argc != 3)
 {
       fprintf(stderr, "Usage: lindist inputFileName outputFileName\n");
       return -1;
 }
 else
 {
      if ( (ingraph = fopen( argv[1], "r" )) == NULL )
        {
           fprintf(stderr, "Cannot open input \
         file.\n");
         return -1;
      } // close if unsuccessful open
      else
      {
         fprintf(stderr, "File %s was opened successfully\n", argv[1] );
        } // end else, this message prints


       if ( (outgraph = fopen( argv[2], "w" )) == NULL )
       {
         fprintf(stderr, "Cannot open output file\n");
```

```
        return -1;
       }
       else
       {
        fprintf(stderr, "File %s was opened successfully\n", argv[2] );
       }
 }  // end argc else

// read the input file into the graph

      fscanf( ingraph, "%d\n", &FILEREADLOOP );
        printf("Diagnostic:  value of FILEREADLOOP is:  %d\n", FILEREADLOOP
);
      NUMNODES = FILEREADLOOP;
      // int linesread = 0;

      for ( i = 0; i < NUMNODES; ++i)
      {
      fscanf( ingraph, "%s%s\n",  g.nodes[i].cellname,
g.nodes[i].lineagestring );
       printf("Diagnostic:  loop %d is now reading node %s\n", i,
g.nodes[i].cellname);
       printf("Diagnostic:  loop %d is now reading node %s\n", i,
g.nodes[i].lineagestring );
      }  // end for read nodes


OLDNUMNODES = NUMNODES;

// close the graph-file
fclose(ingraph);

// Calculate Distances

       printf("Calculating Distances\n");

      i = 0;
      j = 0;

      printf("\nValue of NUMNODES is: %d", NUMNODES);
      for(i = 0; i < NUMNODES; ++i)
            for(j = 0; j < NUMNODES; ++j)
            {
                  g.arcs[i][j].distance =
lineage_distance(g.nodes[i].lineagestring, g.nodes[j].lineagestring);
                  printf("Diag: node1: %s", g.nodes[i].lineagestring);
                  printf("  ");
                  printf("Diag: node1: %s", g.nodes[j].lineagestring);
                  printf("  ");
                  printf("Diag: distance is: %d", g.arcs[i][j].distance);
                  printf("\n");
            }
```

```
      printf("Writing out distance matrix file\n");

g.nodes[i].lineagestring  );
      // }
// Output distances to matrix file


      for ( i = 0; i <  NUMNODES; ++i)
            {
          for (j = 0; j < NUMNODES; ++j)
                {
            fprintf( outgraph, "%d ", g.arcs[i][j].distance );
            printf("Diag:  i = %d, j = %d", i, j);
            }
            fprintf( outgraph, "\n");
          }

      printf("Wrote out distance matrix file\n");
// close the graph-file
fclose(outgraph);
fclose(ingraph);



} // end main
```

5. LISP Program: syndist.lisp()

```
(defun syndist (frame)
    "Take a graph frame, as results, for example, from (load-test-worm), run it through syndist, and get the resulting matrix into the slot #^matrix of the given frame."
    (with-temp-file-in (syndist-input
                common-lisp-user:*tmp-directory* :type
                "sdin")
      (with-temp-file-in (syndist-output
                  common-lisp-user:*tmp-directory* :type
                  "sdout")
        (with-temp-file-in (syndist-log
                    common-lisp-user:*tmp-directory* :type
                    "sdlog")
          (with-open-file (o syndist-input :direction :output
                      :if-exists :supersede)
            (format o "~a~%" (length (#^nodes frame)))
            (loop for node in (#^nodes frame) do
                (format o "~a~%" node))
            (format o "~a~%" (length (#^edges frame)))
            (loop for (from n to) in (#^edges frame) do
                (format o "~a~c~a~c~a~%" from #\Tab to #\Tab n)))
          (block exit
            (excl:run-shell-command
              (print (format nil
                      "cd /home/ericw/c/worm/syndist;./syndist ~a ~a > ~a"
                      syndist-input syndist-output
                      syndist-log)))
            (with-open-file (i syndist-output)
              (setf (#^matrix frame)
                  (loop for node in (#^nodes frame) collect
                      (read-from-string
                        (format
                          nil
                          "(~a)"
                          (read-line i nil nil)))))))
          frame))))
```

6. C++/UNIX Program: SynDistCL().cpp:

```
#include <iostream>
#include <fstream>
#include <set>
#include <string>
#include <list>
#include "Graph.h"  // the graph class
#include "Utilities.h"  // function writeContainer()

using namespace std;

int main(int argc,char *argv[])
{
// graph with vertices of type string
 graph<string> demoGraph;

 demoGraph.clear();

 // input stream for graph data
 ofstream graphOut;
 ifstream graphIn;

 if(argc != 3)
 {
        cout << "Usage:  syndist inputFileName outputFileName" << endl;
        return 0;
 }
 else
 {
        graphIn.open(argv[1]);
        if(!graphIn)
        {
                cout << "Could not open input file!" << endl;
                return 0;
        }
        else
                cout << "Opened input file " << argv[1] << " successfully." << endl;
        graphOut.open(argv[2]);
        if(!graphOut)
        {
                cout << "Could not open output file!" << endl;
                return 0;
        }
        else
                cout << "Opened output file " << argv[2] << " successfully." <<endl;
 }

 // Input the Graph
 graphIn >> demoGraph;
```

```
cout << "Completed Reading Graph" << endl;
cout << "Num Vertices is: " << demoGraph.numberOfVertices() << endl;
cout << "Num Edges is: " << demoGraph.numberOfEdges() << endl;

// Calculate Graph Distances

int i = 0, j = 0;
int value = 0;
int nVertices = demoGraph.numberOfVertices();
list<string> path;

cout << "Starting to calculate distances" << endl;


for(i = 0; i < nVertices; ++i)
{
 for(j = 0; j < nVertices; ++j)
 {
  graphOut << shortestPath(demoGraph, (*(demoGraph.vInfo[i].vtxMapLoc)).first,
  (*(demoGraph.vInfo[j].vtxMapLoc)).first, path);
  cout << "Vertex 1:  " << (*(demoGraph.vInfo[i].vtxMapLoc)).first << " Vertex 2: " <<
  (*(demoGraph.vInfo[j].vtxMapLoc)).first << endl;
  graphOut <<  " ";
  path.clear();
 }
 graphOut << endl;
}

cout << "Done Calculating Distances" << endl;

graphOut.close();
graphIn.close();

return 0;
}
```

10. The UNIX/C++ Program: strongcomponents()

```
#include <iostream>
#include <fstream>
#include <set>
#include <string>
#include <list>
#include "Graph.h"  // the graph class
#include "Utilities.h"  // function writeContainer()

using namespace std;

int main(int argc,char *argv[])
{
// int argc;
// char *argv[3];
 // graph with vertices of type char
 graph<string> demoGraph;

 demoGraph.clear();

 // input stream for graph data
 // ifstream graphIn;
 ofstream graphOut;
 ifstream graphIn;

 if(argc != 3)
 {
        cout << "Usage:  syndist inputFileName outputFileName" << endl;
        return 0;
 }
 else
 {
        graphIn.open(argv[1]);
        if(!graphIn)
        {
                cout << "Could not open input file!" << endl;
                return 0;
        }
        else
                cout << "Opened input file " << argv[1] << " successfully." << endl;
        graphOut.open(argv[2]);
        if(!graphOut)
        {
                cout << "Could not open output file!" << endl;
                return 0;
        }
        else
                cout << "Opened output file " << argv[2] << " successfully." <<endl;
 }

 // Input the Graph
```

```
graphIn >> demoGraph;

cout << "Completed Reading Graph" << endl;
cout << "Num Vertices is: " << demoGraph.numberOfVertices() << endl;
cout << "Num Edges is: " << demoGraph.numberOfEdges() << endl;

// Identify Strong Components


vector<set<string> > vectSet;

strongComponents(demoGraph, vectSet);

for(int i = 0; i < vectSet.size(); i++)
{
        cout << "Strong Component " << i+1 << ":   ";
        writeContainer(vectSet[i].begin(), vectSet[i].end(), " ");
        cout << endl;
}


graphOut.close();
graphIn.close();

return 0;
}
```

6a. Syndist Routine operator>>

```
// input a graph
friend istream& operator>> (istream& istr, graph<T>& g)
{
	// nVertices is number of vertices to read
	int i, nVertices, nEdges;
	// use for input of vertices (v1) and edges ( {v1, v2} )
	T v1, v2;
	// edge weight
	int weight;

	if (g.numVertices > 0)
		// remove an existing graph
		g.clear();

	// input the number of vertices
	istr >> nVertices;

	// input the vertices and insert each into the graph
	for (i = 0; i < nVertices; i++)
	{
		istr >> v1;
		g.insertVertex(v1);
	}

	// input the number of edges
	istr >> nEdges;

	// input the vertices and weight for each edge, and
	// insert it into the graph
	for (i = 0; i < nEdges; i++)
	{
		istr >> v1;
		istr >> v2;
		istr >> weight;
		g.insertEdge(v1,v2,weight);
	}

	// return the stream
	return istr;
}
```

6.a.i. The operator>> routine clear():

```
// erase the graph
template <typename T>
void graph<T>::clear()
{
	// clear the vertex list, vertex map and the
	// availability stack
	vInfo.erase(vInfo.begin(), vInfo.end());
	vtxMap.erase(vtxMap.begin(), vtxMap.end());
	while(!availStack.empty())
		availStack.pop();

	// set the number of vertices and edges to 0
	numVertices = 0;
	numEdges = 0;
}
```

6.a.ii. The operator>> routine: insertVertex():

```
// insert v into the graph
template <typename T>
void graph<T>::insertVertex(const T& v)
{
	int index;

	// attempt to insert v into the map with index 0.
	// if successful, insert an iterator pointing at it
	// into the vertex list at location index. index is obtained
	// from the availability stack or by creating a new entry
	// at the back of the vector. fix the map entry to refer
	// to index and increment numVertices. if the insertion did
	// not take place, the vertex already exists. generate an
	// exception
	pair<vertexMap::iterator, bool> result =vtxMap.insert(vertexMap::value_type(v,0));
	if (result.second)
	{
		// see if there is an entry in vInfo freed by an earlier
		// call to eraseVertex()
		if (!availStack.empty())
		{
			// yes. get its index
			index = availStack.top();
			availStack.pop();
			// call to constructor builds a empty edge set
			vInfo[index] = vertexInfo<T>(result.first);
		}
		else
		{
			// no. we'll have to increase the size of vInfo
			vInfo.push_back(vertexInfo<T>(result.first));
			index = numVertices;
		}

		(*result.first).second = index;
		numVertices++;
	}
	else
		throw graphError("graph insertVertex(): vertex already in the graph");
}
```

6.a.iii. The operator>> routine: insertEdge():

```
// add the edge (v1,v2) with specified weight to the graph
template <typename T>
void graph<T>::insertEdge(const T& v1, const T& v2, int w)
{
	// obtain the vInfo indices
	int pos1=getvInfoIndex(v1), pos2=getvInfoIndex(v2);

	// check for an error
	if (pos1 == -1 || pos2 == -1)
		// if v1 or v2 not in list of vertices, throw an exception
		throw graphError("graph insertEdge(): vertex not in the graph");
	else if (pos1 == pos2)
		// we do not allow self-loops
		throw graphError("graph insertEdge(): self-loops are not allowed");


	// attempt to insert edge (pos2,w) into the edge set of vertex pos1
	pair<set<neighbor>::iterator, bool> result =
				vInfo[pos1].edges.insert(neighbor(pos2,w));

	// make sure edge was not already in the set
	if (result.second)
	{
		// increment the number of edges
		numEdges++;
		// the in-degree of v2 is one more
		vInfo[pos2].inDegree++;
	}
}
```

6.b. The SynDistCL.cpp routine shortestPath():

```
// use the breadth-first traversal algorithm to determine the
// minimum number of edges in any path from sVertex to eVertex
// or -1 if no path exists. if a path exists, the list path
// is the sequence of vertices
friend int shortestPath(graph<T>& g, const T& sVertex,
                                        const T& eVertex, list<T>& path)
{
  // queue stores vertices as vInfo indices
  queue<int>  visitQueue;

  // eIter scans the vertices in an adjacency set
  set<neighbor>::iterator eIter;

        // flag set true when scan identifies eVertex as a neighbor
        bool foundShortestPath = false;

        // index in vInfo for the source and destination vertices
        // and the index for current vertex and a neighbor
        int pos_sVertex, pos_eVertex, currPos, neighborPos;

        int returnValue;

        // initialize all vertices to undiscoved (WHITE)
        for (int i = 0;i < g.vInfo.size(); i++)
                if (g.vInfo[i].occupied)
                        g.vInfo[i].color = vertexInfo<T>::WHITE;

        // obtain the starting and ending indices
        pos_sVertex = g.getvInfoIndex(sVertex);
        pos_eVertex = g.getvInfoIndex(eVertex);

        if (pos_sVertex == -1 || pos_eVertex == -1)
                throw graphError("graph shortestPath(): vertex not in "
                                                "the graph");

        g.vInfo[pos_sVertex].parent = pos_sVertex;
        g.vInfo[pos_sVertex].dataValue = 0;

        // insert starting vertex into the queue
        visitQueue.push(pos_sVertex);

        while (!visitQueue.empty() && !foundShortestPath)
        {
                // delete a queue entry, and color it BLACK
                currPos = visitQueue.front();
                visitQueue.pop();
                g.vInfo[currPos].color = vertexInfo<T>::BLACK;

                // if we are at eVertex, we have found the shortest
                // path from sVertex to eVertex
```

```
                if (currPos == pos_eVertex)
                        foundShortestPath = true;
                else
                {
                        // create an alias for the adjacency set of currPos
                        set<neighbor>& edgeSet = g.vInfo[currPos].edges;

                        // for all undiscovered neighbors, update the dataValue,
                        // color, and parent fields in the vertexInfo object.
                        for (eIter = edgeSet.begin(); eIter != edgeSet.end(); eIter++)
                        {
                                neighborPos = (*eIter).dest;

                                if (g.vInfo[neighborPos].color == vertexInfo<T>::WHITE)
                                {
                                g.vInfo[neighborPos].dataValue = g.vInfo[currPos].dataValue + 1;
                                        g.vInfo[neighborPos].parent = currPos;
                                        g.vInfo[neighborPos].color = vertexInfo<T>::GRAY;
                                        // add neighbor vertex to the queue
                                        visitQueue.push(neighborPos);
                                }
                        }
                }
        }

        // clear path and find the sequence of vertices
        // from sVertex to eVertex
  path.erase(path.begin(), path.end());
  if (foundShortestPath)
        {
                currPos = pos_eVertex;
                while (currPos != pos_sVertex)
                {
                        path.push_front((*(g.vInfo[currPos].vtxMapLoc)).first);
                        currPos = g.vInfo[currPos].parent;
                }
                path.push_front(sVertex);
                returnValue = g.vInfo[pos_eVertex].dataValue;
  }
  else
    returnValue = -1;

        return returnValue;
}
```

6.b.i. ShortestPath() routine: getvInfoIndex():

```
// uses vtxMap to obtain the index of v in vInfo
template <typename T>
int graph<T>::getvInfoIndex(const T& v) const
{
	// iter used in map lookup
	vertexMap::const_iterator iter;
	// index that is returned
	int pos;

	// find the map entry with key v
	iter = vtxMap.find(v);

	// make sure v is in the map
	if (iter == vtxMap.end())
		pos = -1;
	else
		// the index into vInfo is the value of the map entry
		pos = (*iter).second;

	return pos;
}
```

6.c SynDistCL.cpp() header file Graph.h:

This header file defines the graph data structure used in SynDistCL.cpp.

```
#ifndef GRAPH_CLASS
#define GRAPH_CLASS

#include <iostream>
#include <fstream>

#include <set>                          // set class
#include <map>                          // ist classmap class
#include <vector>                       // vector class
#include <list>                 // list class
#include <stack>                // stack class
#include <queue>                        // queue class
#include <functional>   // less<T>

#include "d_except.h"   // exception classes
#include "d_pqueue.h"   // miniPQ class

// largest positive integer on the machine
const int INFINITY = (int)((unsigned int)~0 >> 1);

using namespace std;

class neighbor
{
        public:
                // index of the destination vertex in the vector vInfo of vertex
                // properties
                int dest;
                // weight of this edge
                int weight;

                // constructor
                neighbor(int d=0, int c=0): dest(d), weight(c)
                {}

                // operators for the neighbor class that compare the
                // destination vertices
                friend bool operator< (const neighbor& lhs, const neighbor& rhs)
                {
                        return lhs.dest < rhs.dest;
                }

                friend bool operator== (const neighbor& lhs, const neighbor& rhs)
                {
                        return lhs.dest == rhs.dest;
                }
```

```
};

// maintains vertex properties, including its set of
// neighbors
template <typename T>
class vertexInfo
{
	public:
		// used by graph algorithms
		enum vertexColor { WHITE, GRAY, BLACK };

		// iterator pointing at a pair<T,int> object in the vertex map
		map<T,int>::iterator vtxMapLoc;

		// set of adjacent (neighbor) objects for the current vertex
		set<neighbor> edges;

		/// maintains the in-degree of the vertex
		int inDegree;

		// indicates whether the object currently represents a vertex
		bool occupied;

		// indicate if a vertex is marked in an algorithm that traverses
		// the vertices of a graph
		vertexColor color;

		// available to algorithms for storing relevant data values
		int dataValue;

		// available to graph algorithms; holds parent which is
		// a vertex that has an edge terminating in the current vertex
		int parent;

		// default constructor
		vertexInfo(): inDegree(0), occupied(true)
		{}

		// constructor with iterator pointing to the vertex in the map
		vertexInfo(map<T,int>::iterator iter):
				vtxMapLoc(iter), inDegree(0), occupied(true)
		{}
};

// priority queue data used by minimumPath() and minSpanningTree() algorithms
class minInfo
{
	public:
		int endV;
		int pathWeight;

		friend bool operator< (minInfo lhs, minInfo rhs)
```

```
                { return lhs.pathWeight < rhs.pathWeight; }
};

template <typename T>
class graph
{
  public:
    class const_iterator: public map< T, int >::const_iterator
    {
      public:
        const_iterator()
        {}

                                    // converts a map iterator to a graph iterator
        const_iterator(map<T,int>::const_iterator i)
        {
                                        *((map< T, int >::const_iterator *)this) = i;
                                    }

                                    // return the vertex pointed to by the iterator
        const T& operator* () const
        {
          map<T,int>::const_iterator p = *this;

          return (*p).first;
        }
    };

                typedef const_iterator iterator;

                graph();
                        // constructor. initialize numVertices and numEdges to 0

                graph(const graph<T>& g);
                        // copy constructor

                graph<T>& operator= (const graph<T>& rhs);
                        // overloaded assignment operator

                int numberOfVertices() const;
                        // return the number of vertices in the graph

                int numberOfEdges() const;
                        // return the number of edges in the graph

    bool empty() const;
                        // is the graph empty?

                int getWeight(const T& v1, const T& v2) const;
                        // return the weight of the edge (v1, v2). if the edge.
                        // does not exist, return -1
                        // Precondition: v1 and v2 are vertices in the graph. if not
```

```
                // the function throws the graphError exception

        void setWeight(const T& v1, const T& v2, int w);
                // update the weight of edge (v1, v2).
                // Precondition: v1 and v2 are vertices in the graph. if not,
                // the function throws the graphError exception
                // Postcondition: the weight of vertex (v1,v2) is w

        int inDegree(const T& v) const;
                // return the number of edges entering  v.
                // Precondition: v is a vertex in the graph. if not,
                // the function throws the graphError exception

        int outDegree(const T& v) const;
                // return the number of edges leaving  v.
                // Precondition: v is a vertex in the graph. if not,
                // the function throws the graphError exception

set<T> getNeighbors(const T& v) const;
                // return a set containing the neighbors of v.
                // Precondition: v is a vertex in the graph. if not,
                // the function throws the graphError exception

        void insertEdge(const T& v1, const T& v2, int w);
                // add the edge (v1,v2) with specified weight to the graph.
                // Precondition: v1 and v2 are vertices in the graph. if not,
                // the function throws the graphError exception
                // Postcondition: The number of edges increases by 1

        void insertVertex(const T& v);
                // insert v into the graph.
                // Precondition: v is a vertex in the graph. if not,
                // the function throws the graphError exception.
                // Postcondition: the number of vertices increases by 1

        void eraseEdge(const T& v1, const T& v2);
                // erase edge (v1,v2) from the graph
                // Precondition: v1 and v2 are vertices in the graph. if not,
                // the function throws the graphError exception.
                // Postcondition: The number of edges decreases by 1

void eraseVertex(const T& v);
                // erase v from the graph
                // Precondition: v is a vertex in the graph. if not,
                // the function throws the graphError exception.
                // Postconditions: The number of vertices decreases by 1,
                // and the operation removes all edges to or from v

        void clear();
                // remove all the vertices and edges from the graph

        iterator begin();
```

```
                iterator end();
                const_iterator begin() const;
                const_iterator end() const;
                        // iterator functions returns corresponding map iterator

// "d_galgs.h" implements the graph algorithms using inline code.
// this is necessary for the Borland C++ 5.5 compiler
#include "d_galgs.h"

/*
                LISTING OF THE PROTOTYPES FOR THE GRAPH ALGORITHMS

                friend istream& operator>> (istream& istr, graph<T>& g);
                        // input a graph

                friend ostream& operator<< (ostream& ostr, const graph<T>& g);
                        // output a graph

                friend set<T> bfs(graph<T>& g, const T& sVertex);
                        // perform the breadth-first traversal from sVertex and
                        // return the set of visited vertices

                friend int shortestPath(graph<T>& g, const T& sVertex, const T& eVertex, list<T>& path);
                        // use the breadth-first traversal algorithm to determine the
                        // minimum number of edges in any path from sVertex to eVertex
                        // or -1 if no path exists. if a path exists, the list path
                        // is the sequence of vertices

                friend int minimumPath(graph<T>& g, const T& sVertex, const T& eVertex,
                                                                        list<T>& path);
                        // find the path with minimum total weight from sVertex to eVertex
                        // and return the minimum weight

                friend int minSpanTree(graph<T>& g, graph<T>& MST);
                        // find the minimum spanning tree for the strongly connected digraph g

                friend bool acyclic(graph<T>& g);
                        // determine if the graph is acyclic

                friend void dfsVisit(graph<T>& g, const T& sVertex, list<T>& dfsList,
                                                                        bool checkForCycle);
                        // depth-first visit assuming a WHITE starting vertex. dfsList
                        // contains the visited vertices in reverse order of finishing time.
                        // when checkForCycle is true, the function throws an exception if
                        // it detects a cycle

                friend void dfs(graph<T>& g, list<T>& dfsList);
                        // depth-first search. dfsList contains all the graph vertices in the
                        // reverse order of their finishing times

                friend void topologicalSort(graph<T>& g, list<T>& tlist);
                        // find a topological sort of an acyclic graph
```

```
                friend graph<T> transpose(graph<T>& g);
                        // return the transpose of the graph

                friend void strongComponents(graph<T>& g, vector<set<T> >& component);
                        // find the strong components of the graph
*/

        private:
    typedef map<T,int> vertexMap;

                vertexMap vtxMap;
                        // store vertex in a map with its name as the key and the index
                        // of the corresponding vertexInfo object in the vInfo
                        // vector as the value

                vector<vertexInfo<T> > vInfo;
                        // list of vertexInfo objects corresponding to the vertices

                int numVertices;
                int numEdges;
                        // current size (vertices and edges) of the graph

                stack<int> availStack;
                        // availability stack for storing unused indices in vInfo

                int getvInfoIndex(const T& v) const;
                // uses vtxMap to obtain the index of v in vInfo
};

// uses vtxMap to obtain the index of v in vInfo
template <typename T>
int graph<T>::getvInfoIndex(const T& v) const
{
        // iter used in map lookup
        vertexMap::const_iterator iter;
        // index that is returned
        int pos;

        // find the map entry with key v
        iter = vtxMap.find(v);

        // make sure v is in the map
        if (iter == vtxMap.end())
                pos = -1;
        else
                // the index into vInfo is the value of the map entry
                pos = (*iter).second;

        return pos;
}
```

```
// constructor. initialize numVertices and numEdges to 0
template <typename T>
graph<T>::graph(): numVertices(0), numEdges(0)
{}

// copy constructor
template <typename T>
graph<T>::graph(const graph<T>& g)
{
	*this = g;		// copy g to current object
}

// overloaded assignment operator
template <typename T>
graph<T>& graph<T>::operator= (const graph<T>& rhs)
{
	vertexMap::iterator mi;

	// can't copy a graph to itself
	if (this == &rhs)
		return *this;

	// copy rhs to current object
	vtxMap = rhs.vtxMap;
	vInfo = rhs.vInfo;
	numVertices = rhs.numVertices;
	numEdges = rhs.numEdges;
	availStack = rhs.availStack;

	// update each vtxMapLoc value of objects in vInfo so it points
	// to a key-value pair in the copy of rhs.vtxMap and not rhs.vtxMap
	for (mi=vtxMap.begin();mi != vtxMap.end();mi++)
			vInfo[(*mi).second].vtxMapLoc = mi;

	return *this;
}

// ATTRIBUTE TESTING FUNCTIONS

template <typename T>
int graph<T>::numberOfVertices() const
{
	return numVertices;
}

template <typename T>
int graph<T>::numberOfEdges() const
{
	return numEdges;
}

template <typename T>
```

```
bool graph<T>::empty() const
{
	return numVertices == 0;
}

// ACCESS MEMBER FUNCTIONS

// return the weight of the edge (v1, v2). if the edge
// does not exist, return -1
template <typename T>
int graph<T>::getWeight(const T& v1, const T& v2) const
{
	// find the vInfo indices for the two vertices
	int pos1=getvInfoIndex(v1), pos2=getvInfoIndex(v2);

	// check for an error
	if (pos1 == -1 || pos2 == -1)
		// if v1 or v2 not in list of vertices, throw an exception
		throw graphError("graph getWeight(): vertex not in the graph");

	// construct an alias for the edge list in vInfo[pos1]
	const set<neighbor>& edgeSet = vInfo[pos1].edges;
	set<neighbor>::const_iterator setIter;

	// search for pos2 in the edge list and return its weight
	// if found; otherwise, return -1 to indicate that the
	// edge does not exist
	if ((setIter = edgeSet.find(neighbor(pos2))) != edgeSet.end())
		return (*setIter).weight;
	else
		return -1;
}

template <typename T>
void graph<T>::setWeight(const T& v1, const T& v2, int w)
{
	// find the vInfo indices for the two vertices
	int pos1=getvInfoIndex(v1), pos2=getvInfoIndex(v2);

	// check for an error
	if (pos1 == -1 || pos2 == -1)
		// if v1 or v2 not in list of vertices, throw an exception
		throw graphError("graph setWeight(): vertex not in the graph");

	// construct an alias for the edge list in vInfo[pos1]
	set<neighbor>& edgeSet = vInfo[pos1].edges;
	set<neighbor>::iterator setIter;

	// search for pos2 in the edge list and update its weight.
	// if the edge does not exist, throw an exception
	if ((setIter = edgeSet.find(neighbor(pos2))) != edgeSet.end())
		(*setIter).weight = w;
```

```
	else
		throw graphError("graph setWeight(): edge not in the graph");
}

// return the number of edges entering  v
template <typename T>
int graph<T>::inDegree(const T& v) const
{
	// find the vInfo index for v
	int pos=getvInfoIndex(v);

	if (pos != -1)
		// in-degree is stored in vInfo[pos]
		return vInfo[pos].inDegree;
	else
		// throw an exception
		throw graphError("graph inDegree(): vertex not in the graph");
}

// return the number of edges leaving  v
template <typename T>
int graph<T>::outDegree(const T& v) const
{
	// find the vInfo index for v
	int pos=getvInfoIndex(v);

	if (pos != -1)
		// out-degree is number of elements in the edge set
		return vInfo[pos].edges.size();
	else
		// throw an exception
		throw graphError("graph outDegree(): vertex not in the graph");
}

// return the list of all adjacent vertices
template <typename T>
set<T> graph<T>::getNeighbors(const T& v) const
{
	// set returned
	set<T> adjVertices;

	// obtain the position of v from the map
	int pos = getvInfoIndex(v);

	// if v not in list of vertices, throw an exception
	if (pos == -1)
		throw
			graphError("graph getNeighbors(): vertex not in the graph");

	// construct an alias for the set of edges in vertex pos
	const set<neighbor>& edgeSet = vInfo[pos].edges;
	// use setIter to traverse the edge set
```

```
        set<neighbor>::const_iterator setIter;

        // index of vertexInfo object corresponding to an adjacent vertex
        int aPos;

        for (setIter=edgeSet.begin(); setIter != edgeSet.end(); setIter++)
        {       // "(*setIter).dest" is index into vInfo
                aPos = (*setIter).dest;
                // insert vertex data into a set. vInfo[aPos].vtxMapLoc"
                // is a map iterator. dereference it to access the vertex
                adjVertices.insert ((*vInfo[aPos].vtxMapLoc).first);
        }

  return adjVertices;
}

// GRAPH MODIFICATION MEMBER FUNCTIONS

// add the edge (v1,v2) with specified weight to the graph
template <typename T>
void graph<T>::insertEdge(const T& v1,
                                                        const T& v2, int w)
{
        // obtain the vInfo indices
        int pos1=getvInfoIndex(v1), pos2=getvInfoIndex(v2);

        // check for an error
        if (pos1 == -1 || pos2 == -1)
                // if v1 or v2 not in list of vertices, throw an exception
                throw graphError("graph insertEdge(): vertex not in the graph");
        else if (pos1 == pos2)
                // we do not allow self-loops
                throw graphError("graph insertEdge(): self-loops are not allowed");


        // attempt to insert edge (pos2,w) into the edge set of vertex pos1
        pair<set<neighbor>::iterator, bool> result =
                        vInfo[pos1].edges.insert(neighbor(pos2,w));

        // make sure edge was not already in the set
        if (result.second)
        {
                // increment the number of edges
                numEdges++;
                // the in-degree of v2 is one more
                vInfo[pos2].inDegree++;
        }
}

// insert v into the graph
template <typename T>
void graph<T>::insertVertex(const T& v)
```

```
{
        int index;

        // attempt to insert v into the map with index 0.
        // if successful, insert an iterator pointing at it
        // into the vertex list at location index. index is obtained
        // from the availability stack or by creating a new entry
        // at the back of the vector. fix the map entry to refer
        // to index and increment numVertices. if the insertion did
        // not take place, the vertex already exists. generate an
        // exception
        pair<vertexMap::iterator, bool> result =
                vtxMap.insert(vertexMap::value_type(v,0));
        if (result.second)
        {
                // see if there is an entry in vInfo freed by an earlier
                // call to eraseVertex()
                if (!availStack.empty())
                {
                        // yes. get its index
                        index = availStack.top();
                        availStack.pop();
                        // call to constructor builds a empty edge set
                        vInfo[index] = vertexInfo<T>(result.first);
                }
                else
                {
                        // no. we'll have to increase the size of vInfo
                        vInfo.push_back(vertexInfo<T>(result.first));
                        index = numVertices;
                }

                (*result.first).second = index;
                numVertices++;
        }
        else
                throw graphError("graph insertVertex(): vertex already in the graph");
}

// erase edge (v1,v2) from the graph
template <typename T>
void graph<T>::eraseEdge(const T& v1, const T& v2)
{
        // obtain the indices of v1 and v2 in vInfo
        int pos1=getvInfoIndex(v1), pos2=getvInfoIndex(v2);

        // check for an error
        if (pos1 == -1 || pos2 == -1)
                // if v1 or v2 not in list of vertices, throw an exception
                throw graphError("graph eraseEdge(): vertex not in the graph");

        // construct an alias to the edge set of vInfo[pos1]
```

```
        set<neighbor>& edgeSet = vInfo[pos1].edges;
        set<neighbor>::iterator setIter;

        // search for pos2 in the edge set
        setIter = edgeSet.find(neighbor(pos2));
        // if pos2 is in the set, erase it; otherwise, output an
        // error message
        if (setIter != edgeSet.end())
        {
                edgeSet.erase(setIter);
                // the in-degree of v2 is one less
                vInfo[pos2].inDegree--;
                numEdges--;
        }
        else
                throw graphError("graph eraseEdge(): edge not in the graph");
}

template <typename T>
void graph<T>::eraseVertex(const T& v)
{
        // use to search for and remove v from the map
        vertexMap::iterator mIter;
        // pos is index of v in the vertex list
        int pos, j;
        // used in removal of edges to v
        set<neighbor>::iterator sIter;

        // search the map for the key v
        mIter = vtxMap.find(v);
        // if vertex is not present, terminate the erase
        if (mIter == vtxMap.end())
                // if v not in list of vertices, throw an exception
                throw graphError("graph eraseVertex(): vertex not in the graph");

        // obtain the index of v in vInfo
        pos = (*mIter).second;

        // erase vertex from the map and decrement number of vertices
        vtxMap.erase(mIter);
        numVertices--;

        // mark the vertex entry in vInfo as not occupied. the index pos is now
        // available. push it on the availability stack for use later if we
        // insert a vertex
        vInfo[pos].occupied = false;
        availStack.push(pos);

        // cycle through vInfo and remove all edges going to v
        for (j=0; j < vInfo.size(); j++)
                // skip all unoccupied entries, including pos
                if (vInfo[j].occupied)
```

```
                {
                        // construct an alias for the set vInfo[j].edges
                        set<neighbor>& edgeSet = vInfo[j].edges;

                        sIter = edgeSet.begin();
                        // cycle through the edge set
                        while (sIter != edgeSet.end())
                                if ((*sIter).dest == pos)
                                {
                                        // found pos. remove it from the set and
                                        // decrement the edge count
                                        edgeSet.erase(sIter);
                                        numEdges--;
                                        break;
                                }
                                else
                                        // took no action. just move forward
                                        sIter++;
                }

        // decrement numEdges by the number of edges for vertex v
        numEdges -= vInfo[pos].edges.size();

        // the in-degree for all of v's neighbors must be decreased by 1
        set<neighbor>& edgesFromv = vInfo[pos].edges;
        for (sIter=edgesFromv.begin(); sIter != edgesFromv.end(); sIter++)
        {
                j = (*sIter).dest;
                vInfo[j].inDegree--;
        }

        // clear the edge set. construct an alias for vInfo[pos].edges
        // and use erase to clear the set
        set<neighbor>& edgeSet = vInfo[pos].edges;
        edgeSet.erase(edgeSet.begin(), edgeSet.end());
}

// erase the graph
template <typename T>
void graph<T>::clear()
{
        // clear the vertex list, vertex map and the
        // availability stack
        vInfo.erase(vInfo.begin(), vInfo.end());
        vtxMap.erase(vtxMap.begin(), vtxMap.end());
        while(!availStack.empty())
                availStack.pop();

        // set the number of vertices and edges to 0
        numVertices = 0;
        numEdges = 0;
}
```

```
// ITERATOR FUNCTIONS

// each graph iterator function returns
// the corresponding map iterator
template <typename T>
graph<T>::iterator graph<T>::begin()
{
  return graph<T>::iterator(vtxMap.begin());
}

template <typename T>
graph<T>::iterator graph<T>::end()
{
        return graph<T>::iterator(vtxMap.end());
}

template <typename T>
graph<T>::const_iterator graph<T>::begin() const
{
        return graph<T>::iterator(vtxMap.begin());
}

template <typename T>
graph<T>::const_iterator graph<T>::end() const
{
  return graph<T>::iterator(vtxMap.end());
}

#endif    // GRAPH_CLASS
```

7. Biobike LISP program: fcor.lisp():

```
(defun fcor (frame-x frame-y)
      "Runs syndist on two frames, then appends together the resulting matrix
and runs a correlation on them."
      (flet ((append-matrix (frame)
               (loop for row in (#^matrix frame) append row)))
        (correlate (append-matrix frame-x) (append-matrix frame-y))))
```

8. C++/UNIX Program newlineage.cpp():

```
#include "Hash.h"
#include "HashFunctions.h"
#include "Vector.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

bool find(string x, vector<string> v)
{
	int i = 0;
	for(i = 0; i < v.size(); ++i)
		if(v[i] == x)
		{
			cout << "found " << x << endl;
			return true;
		}
		else if(i == v.size()-1)
		{
			cout << "didn't find "  << x << endl;
			return false;
		}
}


int main(int argc, char *argv[])
{

	// Open three files
	ifstream newnodes;
	ifstream alllineage;
	ofstream newlineage;

	if(argc != 4)
	{
		cout << "Usage: newlineage newnodelist masterlineagefile newlineagefile" << endl;
		return 0;
	}
	else
	{
		newnodes.open(argv[1]);
		if(!newnodes)
		{
			cout << "Could not open newnodelist file!" << endl;
			return 0;
		}
		else
			cout << "Opened input file " << argv[1] << " successfully." << endl;
		if(!alllineage)
		{
```

```
                cout << "Could not open masterlineagefile file!" << endl;
                return 0;
        }
        else
                cout << "Opened input file " << argv[2] << " successfully." << endl;
        if(!newlineage)
        {
                cout << "Could not open newlineagefile file!" << endl;
                return 0;
        }
        else
                cout << "Opened input file " << argv[3] << " successfully." << endl;

}



// Read new nodes into vector

int numnodes = 0, i = 0;
string temp;
newnodes >> numnodes;
vector<string> v(numnodes);

for(i = 0; i < numnodes; ++i)
{
        newnodes >> temp;
        cout << "Value of temp is: " << temp << endl;
        v.push_back(temp);
        cout << "Value of element in vector is: " << v.back() << endl;
}


// algorithm to write new edges to output file

int numedges = 0, weight;
string node1; string lineage;
alllineage >> numedges;
for(i = 0; i < numedges; ++i)
{
        alllineage >> node1 >> lineage;

        if(find(node1, v))
        {
                newlineage << node1 << " " << lineage << endl;
                cout << "New lineage string written." << endl;

        }
        else
                continue;
}
```

```
        newnodes.close();
        alllineage.close();
        newlineage.close();


        return 0;
}
```

9. UNIX/C++ Program newedges.cpp():

```
#include "Hash.h"
#include "HashFunctions.h"
#include "Vector.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

bool find(string x, vector<string> v)
{
	int i = 0;
	for(i = 0; i < v.size(); ++i)
		if(v[i] == x)
		{
			cout << "found " << x << endl;
			return true;
		}
		else if(i == v.size()-1)
		{
			cout << "didn't find "  << x << endl;
			return false;
		}
}


int main(int argc, char *argv[])
{

	// Open three files
	ifstream newnodes;
	ifstream alledges;
	ofstream newedges;

	// newnodes.open("LeftNeurons.txt");
	// alledges.open("MiniEdges.txt");
	// newedges.open("LeftEdges.txt");

	if(argc != 4)
	{
		cout << "Usage: NewEdges newnodelist masteredgefile newedgefile" << endl;
		return 0;
	}
	else
	{
		newnodes.open(argv[1]);
		if(!newnodes)
```

```
	{
		cout << "Could not open newnodelist file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[1] << " successfully." << endl;
	if(!alledges)
	{
		cout << "Could not open masteredgefile file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[2] << " successfully." << endl;
	if(!newedges)
	{
		cout << "Could not open newedgefile file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[3] << " successfully." << endl;

}



// Read new nodes into vector

int numnodes = 0, i = 0;
string temp;
newnodes >> numnodes;
vector<string> v(numnodes);

for(i = 0; i < numnodes; ++i)
{
	newnodes >> temp;
	cout << "Value of temp is: " << temp << endl;
	v.push_back(temp);
	cout << "Value of element in vector is: " << v.back() << endl;
}


// algorithm to write new edges to output file

int numedges = 0, weight;
string node1; string node2;
alledges >> numedges;
for(i = 0; i < numedges; ++i)
{
	alledges >> node1 >> node2 >> weight;

	if(find(node1, v) && find(node2, v))
```

```
                {
                        newedges << node1 << " " << node2 << " " << weight << endl;
                        cout << "New edge written." << endl;

                }
                else
                        continue;
        }

        newnodes.close();
        alledges.close();
        newedges.close();


        return 0;
}
```

10. The UNIX/C++ Program that calls strongComponent: strongcomponent.cpp():

```
#include <iostream>
#include <fstream>
#include <set>
#include <string>
#include <list>
#include "Graph.h"  // the graph class
#include "Utilities.h"  // function writeContainer()

using namespace std;

int main(int argc,char *argv[])
{
// graph with vertices of type string
 graph<string> demoGraph;

 demoGraph.clear();

 // input stream for graph data
 // ifstream graphIn;
 ofstream graphOut;
 ifstream graphIn;

 if(argc != 3)
 {
        cout << "Usage:  strongcomponents inputFileName outputFileName" << endl;
        return 0;
 }
 else
 {
        graphIn.open(argv[1]);
        if(!graphIn)
        {
                cout << "Could not open input file!" << endl;
                return 0;
        }
        else
                cout << "Opened input file " << argv[1] << " successfully." << endl;
        graphOut.open(argv[2]);
        if(!graphOut)
        {
                cout << "Could not open output file!" << endl;
                return 0;
        }
        else
                cout << "Opened output file " << argv[2] << " successfully." <<endl;
 }

 // Input the Graph
```

```
graphIn >> demoGraph;

cout << "Completed Reading Graph" << endl;
cout << "Num Vertices is: " << demoGraph.numberOfVertices() << endl;
cout << "Num Edges is: " << demoGraph.numberOfEdges() << endl;

// Identify Strong Components


vector<set<string> > vectSet;

strongComponents(demoGraph, vectSet);

for(int i = 0; i < vectSet.size(); i++)
{
        cout << "Strong Component " << i+1 << ":   ";
        writeContainer(vectSet[i].begin(), vectSet[i].end(), " ");
        cout << endl;
}


graphOut.close();
graphIn.close();

return 0;
}
```

10. a. The Routine ‘strongComponent()’:

```
friend void strongComponents(graph<T>& g, vector<set<T> >& component)
{
	// list of vertices visited by dfs() for graph g
	list<T> dfsGList;

	// list of vertices visited by dfsVisit() for g transpose
	list<T> dfsGTList;

	// used to scan dfsGList and dfsGTList objects
	list<T>::iterator gIter, gtIter;

	// transpose of the graph
	graph<T> gt;

	// set for an individual strong component
	set<T> scSet;

	int i;

	// clear the return vector
	component.resize(0);

	// execute depth-first tranversal of g
	dfs(g, dfsGList);

	// compute gt
	gt = transpose(g);

	// initialize all vertices in gt to WHITE (unvisited)
	for (i=0;i < gt.vInfo.size(); i++)
		if (gt.vInfo[i].occupied)
			gt.vInfo[i].color = vertexInfo<T>::WHITE;

	// call dfsVisit() for gt from vertices in dfsGList
	gIter = dfsGList.begin();
	while(gIter != dfsGList.end())
	{
		// call dfsVisit() only if vertex has not been visited
		if (gt.vInfo[gt.getvInfoIndex(*gIter)].color ==vertexInfo<T>::WHITE)
		{
			// clear dfsGTList and scSet
			dfsGTList.erase(dfsGTList.begin(), dfsGTList.end());
			scSet.erase(scSet.begin(), scSet.end());

			// do dfsVisit() in gt for starting vertex *gIter
			dfsVisit(gt, *gIter, dfsGTList, false);

			// copy vertices from the list to set scSet
			for (gtIter = dfsGTList.begin(); gtIter != dfsGTList.end(); gtIter++)
				scSet.insert(*gtIter);
```

```
                    // add strong component set to the vector
                    component.push_back(scSet);
                }
                gIter++;
            }
        }
```

10. a. i. StrongComponent() routine dfsVisit():

```
// depth-first visit assuming a WHITE starting vertex. dfsList
// contains the visited vertices in reverse order of finishing time.
// when checkForCycle is true, the function throws an exception if
// it detects a cycle
friend void dfsVisit(graph<T>& g, const T& sVertex, list<T>& dfsList,
                                                        bool checkForCycle)
{
        // indices for vertex positions in vInfo
        int pos_sVertex, pos_neighbor;

        // iterator to scan the adjacency set of a vertex
        set<neighbor>::iterator adj;

        // alias to simplify access to the vector vInfo
        vector<vertexInfo<T> >& vlist = g.vInfo;

        // fetch the index for sVertex in vInfo; throw an exception
        // if the starting vertex is not in the graph
        pos_sVertex = g.getvInfoIndex(sVertex);

        if (pos_sVertex == -1)
                throw graphError("graph dfsVisit(): vertex not in the graph");

        // color vertex GRAY to note its discovery
        vlist[pos_sVertex].color = vertexInfo<T>::GRAY;

        // create an alias for the adjacency set of sVertex
        set<neighbor>& edgeSet = vlist[pos_sVertex].edges;

        // sequence through the adjacency set and look for vertices
        // that are not yet discovered (colored WHITE). recursively call
        // dfsVisit() for each such vertex. if a vertex in the adjacency
        // set is GRAY, the vertex was discovered during a previous
        // call and there is a cycle that begins and ends at the
        // vertex; if checkForCycle is true, throw an exception
        for (adj = edgeSet.begin(); adj != edgeSet.end(); adj++)
        {
                pos_neighbor = (*adj).dest;
                if (vlist[pos_neighbor].color == vertexInfo<T>::WHITE)
                        dfsVisit(g,(*(g.vInfo[pos_neighbor].vtxMapLoc)).first,
                                                        dfsList, checkForCycle);
                else if (vlist[pos_neighbor].color == vertexInfo<T>::GRAY
                                        && checkForCycle)
                        throw graphError("graph dfsVisit(): graph has a cycle");
        }

        // finished with vertex sVertex. make it BLACK and add it to
        // the front of dfsList
        vlist[pos_sVertex].color = vertexInfo<T>::BLACK;
        dfsList.push_front((*(g.vInfo[pos_sVertex].vtxMapLoc)).first);
```

```
}
```

11. neuronpaths() [C++/UNIX program]

```
#include "Graph.h"
#include "Vector.h"
#include "Utilities.h" // writeContainer()
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <list>


template<typename T>
int find(graph<T> &g, string x)
{
	int r = 0;

	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
			return -1;
		}
}


int main(int argc, char *argv[])
{

	// Open four files
	ifstream graphname;
	ifstream inputneurons;
	ifstream outputneurons;
	ofstream neuronpaths;

	if(argc != 5)
	{
		cout << "Usage: neuronpaths graphname inputneurons outputneurons neuronpathlist "
		<<  endl;
		return 0;
	}
	else
	{
		graphname.open(argv[1]);
		if(!graphname)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
```

```
	}
	else
		cout << "Opened graph file " << argv[1] << " successfully." << endl;
	inputneurons.open(argv[2]);
	if(!inputneurons)
	{
		cout << "Could not open input neuron file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[2] << " successfully." << endl;
	outputneurons.open(argv[3]);
	if(!outputneurons)
	{
		cout << "Could not open output neuron file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[3] << " successfully." << endl;
	neuronpaths.open(argv[4]);
	if(!neuronpaths)
	{
		cout << "Could not open neuron path output file!" << endl;
		return 0;
	}
	else
		cout << "Opened output file " << argv[4] << " successfully." << endl;


}



// Load the input neuron vector

int numinputnodes = 0, numoutputnodes = 0, i = 0;

string temp;

inputneurons >> numinputnodes;

string vinputneurons[numinputnodes];

for(i = 0; i < numinputnodes; ++i)
{
	inputneurons >> temp;
	vinputneurons[i] = temp;
}

// load the output neuron vector
```

```
	outputneurons >> numoutputnodes;

	string voutputneurons[numoutputnodes];

	for(i = 0; i < numoutputnodes; ++i)
	{
		outputneurons >> temp;
		voutputneurons[i] = temp;
	}

	// Open graph
	graph<string> g;
	g.clear();

	graphname >> g;


	// calculate and write out paths from input neurons to output neurons

	list<string> path;

	int p=0;
	int q=0;
	for(p = 0; p < numinputnodes; ++p)
		for(q = 0; q < numoutputnodes; ++q)
		{
			cout << "Begining neuron is: " << vinputneurons[p] << endl;
			cout << "Ending neuron is: " << voutputneurons[q] << endl;
			 cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g,
vinputneurons[p])].vtxMapLoc)).first, (*(g.vInfo[find(g, voutputneurons[q])].vtxMapLoc)).first, path) << endl;
			cout << "Path is: ";
			writeList(path, "  ");
			cout << endl;
		}


	graphname.close();
	inputneurons.close();
	outputneurons.close();
	neuronpaths.close();


	return 0;
}
```

12. neuronenv()

```
#include "Vector.h"
#include "Utilities.h" // writeContainer()
#include "Graph.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <list>

template<typename T>
bool find(list<T> l, string x)
{
	list<T>::iterator liter;
	liter = l.begin();
	int counter = 0;
	while(liter != l.end())
	{
		++counter;
		if(*liter == x)
			return true;
		else if(counter == l.size())
			return false;
		++liter;
	}
}

template<typename T>
int find(graph<T> &g, string x)
{
	int r = 0;
	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
			return -1;
		}
}


int main(int argc, char *argv[])
{
```

```
// Open four files
ifstream graphname;
ofstream neuroncircuit;
string inputneuron;
string outputneuron;

if(argc != 5)
{
	cout << "Usage: neuronenv graphname inputneuron outputneuron neuroncircuit " << endl;
	return 0;
}
else
{
	graphname.open(argv[1]);
	if(!graphname)
	{
		cout << "Could not open graph file!" << endl;
		return 0;
	}
	else
		cout << "Opened graph file " << argv[1] << " successfully." << endl;
	neuroncircuit.open(argv[4]);
	if(!neuroncircuit)
	{
		cout << "Could not open output file!" << endl;
		return 0;
	}
	else
		cout << "Opened output file " << argv[4] << "successfully." << endl;
	inputneuron = argv[2];
	if(isdigit(inputneuron[0]))
	{
		cout << "Second parameter must be a string: neuron name!" << endl;
		return 0;
	}
	else
		cout << "Second parameter is correct." << endl;
	outputneuron = argv[3];
	if(isdigit(outputneuron[0]))
	{
		cout << "Third parameter must be a string:  a neuron name" << endl;
		return 0;
	}
	else
		cout << "Third parameter is correct." << endl;
}

// Open graph
graph<string> g;
g.clear();

graphname >> g;
```

```
	// calculate and write out paths from input neurons to output neurons

	list<string> path;

				cout << "Begining neuron is: " << inputneuron << endl;
				cout << "Ending neuron is: " << outputneuron << endl;
				 cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g,
inputneuron)].vtxMapLoc)).first, (*(g.vInfo[find(g, outputneuron)].vtxMapLoc)).first, path) << endl;
				cout << "Path is: ";
				writeList(path, "  ");
				cout << endl;

	int numvertices = 0;
	int numedges = 0;
	int i = 0, weight = 0;
	string temp, neuron1, neuron2;
	graphname.clear();
	graphname.seekg(0);
	graphname >> numvertices;
	for(i=0; i < numvertices; ++i)
		graphname >> temp;
	graphname >> numedges;

    for(i=0; i < numedges; ++i)
	{
		graphname >> neuron1;
		graphname >> neuron2;
		graphname >> weight;

		if(find(path, neuron1) || find(path, neuron2))
			neuroncircuit << neuron1 << " " << neuron2 << " " << weight << endl;
	}

	graphname.close();
	neuroncircuit.close();

	return 0;
}
```

13. neuronenvgenes()

```
#include <set>
#include <list>
#include <vector>
#include <fstream>
#include <iostream>
#include "d_graph.h"
#include "d_util.h"
#include <string>
#include <algorithm>

// neuronenvgenes is a program which returns the union and intersection of
// genes that are part of a neuron circuit.  A neuron circuit is here defined
// as the set of neurons which impinge upon (have inputs to or outputs from)
// a set of neurons that are part of a path.


class neuron{
	private:
		string name;
		string lineage;
		string maintype;
		string subtype1;
		string subtype2;
		set<string> genes;

	public:
		neuron()
		{}
		void setName(string x)
		{
			name = x;
		}
		void setLineage(string x)
		{
			lineage = x;
		}
		void setMainType(string x)
		{
			maintype = x;
		}
		void setSubType1(string x)
		{
			subtype1 = x;
		}
		void setSubType2(string x)
		{
			subtype2 = x;
		}
```

```
		void addGene(string gene)
		{
			genes.insert(gene);
		}
		string getName()
		{
			return name;
		}
		string getLineage()
		{
			return lineage;
		}
		string getMainType()
		{
			return maintype;
		}
		string getSubType1()
		{
			return subtype1;
		}
		string getSubType2()
		{
			return subtype2;
		}
		set<string> getGenes()
		{
			return genes;
		}
	};


// SET FUNCTION IMPLEMENTATIONS

template <typename T>
bool operator== (const set<T>& lhs, const set<T>& rhs)
{
	typename set<T>::const_iterator myself = lhs.begin(), other = rhs.begin();

	// return false if the sets do not have the same size
	if (lhs.size() == rhs.size())
	{
		// compare until encounter end of the sets or
		// find two elements that are not equal
		while (myself != lhs.end() && *myself++ == *other++);

		// if we left the loop before reaching the end
		// of the sets, they are not equal
		if (myself != lhs.end())
			return false;
		else
			return true;
	}
```

```
	else
		return false;
}

template <typename T>
set<T> operator+ (const set<T>& lhs, const set<T>& rhs)
{
	// constuct union
	set<T> setUnion;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*rhsIter++);
		else
		{
			// the two values are equal. insert just one and
			// move both itertors forward
			setUnion.insert(*lhsIter++);
			rhsIter++;
		}

		// flush any remaining items
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setUnion.insert(*lhsIter++);
		else if (rhsIter != rhs.end())
			while (rhsIter != rhs.end())
				setUnion.insert(*rhsIter++);

	return setUnion;
}

template <typename T>
set<T> operator* (const set<T>& lhs, const set<T>& rhs)
{
	// constuct intersection
	set<T> setIntersection;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
```

```
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter is in lhs and not in rhs. move iterator
			// forward
			lhsIter++;
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in rhs and not in lhs. move iterator
			// forward
			rhsIter++;
		else
		{
			// the same value is in both sets. insert one value
			// and move the iterators forward
			setIntersection.insert(*lhsIter);
			lhsIter++;
			rhsIter++;
		}

		return setIntersection;
}

template <typename T>
set<T> operator- (const set<T>& lhs, const set<T>& rhs)
{
	// construct difference
	set<T> setDifference;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to lhs but not to rhs. put it in
			// the difference
			setDifference.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in the rhs but not in the lhs. pass
			// over it
			rhsIter++;
		else
		{
			// the same value is in both sets. move the
			// iterators forward
			lhsIter++;
			rhsIter++;
		}

		// flush any remaining items from lhs
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setDifference.insert(*lhsIter++);
```

```
	return setDifference;
}

template <typename T>
list<T> vector2list(vector<T> v)
{
	list<T> l;
	for(int i=0; i < v.size(); i++)
		l.push_back(v[i]);
	return l;
}

template <typename T>
vector<T> list2vector(list<T> l)
{
	vector<T> v;
	typename list<T>::iterator liter = l.begin();
	while(liter != l.end())
		v.push_back(*liter++);
	return v;
}

template <typename T>
int find(graph<T> &g, string x)
{
	int r = 0;
	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
			return -1;
		}
}

template <typename T>
bool find(list<T> l, string x)
{
	typename list<T>::iterator liter;
	liter = l.begin();
	int counter = 0;
	while(liter != l.end())
	{
		++counter;
		if(*liter == x)
			return true;
		else if(counter == l.size())
			return false;
```

```
                ++liter;
        }
}

set<string> findGeneSet(string neuronname, neuron neuron[], int numneurons)
{
        for(int i=0; i < numneurons; ++i)
                if(neuronname == neuron[i].getName())
                        return neuron[i].getGenes();
}

int main(int argc, char *argv[])
{
        // cout << "Entered main" << endl;
        ifstream neurons;
        string inputneuron, outputneuron;


        if(argc != 4)
        {
                cout << "Usage: neuroncircuitgenes graphname inputneuron outputneuron " << endl;
                return 0;
        }
        else
        {
                neurons.open(argv[1]);
                if(!neurons)
                {
                        cout << "Could not open graph file!" << endl;
                        return 0;
                }
                else
                        cout << "Opened graph file " << argv[1] << " successfully." << endl;
                inputneuron = argv[2];
                if(isdigit(inputneuron[0]))
                {
                        cout << "Second parameter must be a string: neuron name!" << endl;
                        return 0;
                }
                else
                        cout << "Second parameter is correct." << endl;
                outputneuron = argv[3];
                if(isdigit(outputneuron[0]))
                {
                        cout << "Third parameter must be a string:  a neuron name" << endl;
                        return 0;
                }
                else
                        cout << "Third parameter is correct." << endl;
        }
```

```
	int numneurons = 0;
	string gene;
	vector<neuron> vneurons(numneurons);
	neurons >> numneurons;
	string temp;
	neuron n[numneurons];
	for(int i = 0; i < numneurons; ++i)
	{
		neurons >> temp;
		n[i].setName(temp);
		neurons >> temp;
		n[i].setLineage(temp);
		neurons >> temp;
		n[i].setMainType(temp);
		neurons >> temp;
		n[i].setSubType1(temp);
		neurons >> temp;
		n[i].setSubType2(temp);
		while(1)
		{
			neurons >> gene;
				if(gene == "end")
					break;
			n[i].addGene(gene);
		}
	}
	// insert vertices
	graph<string> g;

	int vneuronsize = vneurons.size();

	for(int i=0; i < numneurons; ++i)
		g.insertVertex(n[i].getName());

	// cout << "print out vertices" << endl;

	// cout << "Number of vertices is: " << g.numVertices << endl;

	//for(int i=0; i < g.numVertices; ++i)
	//	cout << "Vertex is: " << (*(g.vInfo[i].vtxMapLoc)).first << endl;

	int numedges = 0;
	string neuron1, neuron2;
	int weight = 0;

	neurons >> numedges;
//cout << "numedges is: " << numedges << endl;


	struct synapse {
		string n1;
```

```
		string n2;
		int w;
	} synapses[numedges];


	for(int i = 0; i < numedges; ++i)
	{
		neurons >> neuron1;
		synapses[i].n1 = neuron1;
		//cout << "neuron1 is: " << neuron1 << endl;
		neurons >> neuron2;
		synapses[i].n2 = neuron2;
		//cout << "neuron2 is: " << neuron2 << endl;
		neurons >> weight;
		synapses[i].w = weight;
		//cout << "weight is: " << weight << endl;

		g.insertEdge(neuron1, neuron2, weight);
		//cout << "Inserted edge " << i << endl;
	}
	//cout << "Done inserting edges" << endl;



	list<string> path;

	cout << "Begining neuron is: " << inputneuron << endl;
	cout << "Ending neuron is: " << outputneuron << endl;
	cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g, inputneuron)].vtxMapLoc)).first,
(*(g.vInfo[find(g, outputneuron)].vtxMapLoc)).first, path) << endl;
	cout << "Path is: ";
	writeList(path, "  ");
	cout << endl;

	cout << "Creating neuronlist" << endl;
	vector<string> neuronlist;
	cout << "Done Creating neuronlist" << endl;

	vector<set<string> > vs;

	list<string>::iterator liter;
	liter = path.begin();
	set<string>::iterator fsiter;
	set<string>::iterator esiter;

	cout << "Creating neuronlist." << endl;

	while(liter != path.end())
		neuronlist.push_back(*liter++);

	// Now find all neurons that impinge upon neurons in the path.
```

```
cout << "Putting impinging neurons on list" << endl;

for(int i=0; i < numedges; ++i)
{
	if(find(path, synapses[i].n1) || find(path, synapses[i].n2))
	{
		neuronlist.push_back(synapses[i].n1);
		neuronlist.push_back(synapses[i].n2);
	}
}

list<string> neuronlisttemp = vector2list(neuronlist);

// Unique only works if you do sort BEFORE unique.

neuronlisttemp.sort();

neuronlisttemp.unique();

vector<string> neuronlist2 = list2vector(neuronlisttemp);

cout << "Creating vector of gene sets for neurons in circuit" << endl;

for(int i = 0; i < neuronlist2.size(); ++i)
{
	vs.push_back(findGeneSet(neuronlist2[i], n, numneurons));
	cout << endl << endl << "Neuron: " << neuronlist2[i] << " has the following genes: " << endl;
	fsiter = vs.back().begin();
	esiter = vs.back().end();
	writeContainer(fsiter, esiter, " ");
				++liter;
}


cout << endl << endl;
cout << "Genes involved in the neuron circuit that stems from the pathway: ";
writeList(path, " ");
cout << "are: " << endl << endl;

set<string> us;

us = vs[0];

for(int i=0; i < vs.size(); ++i)
	us = us + vs[i];

fsiter = us.begin();
esiter = us.end();
writeContainer(fsiter, esiter, " ");
if(us.empty())
	cout << "There are no genes involved in this pathway in our database." << endl;
```

```
	set<string> is;

	is = vs[0];

	for(int i = 1; i < vs.size(); ++i)
	{
		is = is * vs[i];
	}

	fsiter = is.begin();
	esiter = is.end();
	cout << endl << endl << "The genes that are common to every neuron in this pathway are: " << endl;
	writeContainer(fsiter, esiter, " ");
	if(is.empty())
		cout << "There are no genes common to every neuron in this pathway according to our database."
<< endl;


	return 0;
}
```

14. subpaths()

```
#include "Graph.h"
#include "Vector.h"
#include "Utilities.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <list>


bool sublist(list<string> lsubpath, list<string> lpath)
{
	int subpathlength = lsubpath.size();
	int pathlength = lpath.size();
	int count = 0;

	vector<string> subpath;
	vector<string> path;
	list<string>::iterator spiter;
	spiter = lsubpath.begin();
	list<string>::iterator piter;
	piter = lpath.begin();

	for(int i=0; i < subpathlength; ++i)
	{
		subpath.push_back(*spiter++);
	}
	for(int i=0; i < pathlength; ++i)
	{
		path.push_back(*piter++);
	}
	for(int i = 1; i < pathlength-subpathlength; ++i)
	{
		for(int j = 0; j < subpathlength; ++j)
			if(subpath[j] == path[i+j])
				++count;
		if(count == subpathlength)
			return true;
		count = 0;
	}
	return false;
}

template<typename T>
int find(graph<T> &g, string x)
{
	int r = 0;
	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
```

```
                return r;
        }
        else if(r == g.numVertices - 1)
        {
                cout << "Couldn't find vertex." << endl;
                return -1;
        }
}

bool operator== (list<string> l1, list<string> l2)
{
        if(l1.size() != l2.size())
                return false;
        list<string>::iterator liter1, liter2;
        liter1 = l1.begin();
        liter2 = l2.begin();

        while(liter1 != l1.end())
        {
                if(*liter1 != *liter2)
                        return false;
                else if (liter1 == l1.end())
                        return true;
                ++liter1;
                ++liter2;
        }
}

bool find(list<string> l1, vector<list<string> >& subpaths)
{
        if(subpaths.empty())
                return false;
        for(int i =0; i < subpaths.size(); ++i)
                if(subpaths[i] == l1))
                        return true;
                else if(i == subpaths.size()-1)
                        return false;
}

void nrinsert(list<string> x, vector<list<string> >& subpaths)
{
        if(find(x,subpaths))
                return;
        else
                subpaths.push_back(x);
}

void subpaths(list<string> path, vector<list<string> >& vsubpaths)
{
        vector<string> v;
        list<string>::iterator liter;
        liter = path.begin();
```

```
	for(int i = 0; i < path.size(); ++i)
		v.push_back(*liter++);

	list<string> subpath;
	int passes = v.size()-2;
	int number = v.size()-1;
	int length = 1;
	for(int pass = 0; pass < passes; ++pass)
	{
		for(int i=1; i < number; ++i)
		{
			for(int j=0; j<length; ++j)
				subpath.push_back(v[i+j]);
			nrinsert(subpath, vsubpaths);
			subpath.erase(subpath.begin(), subpath.end());
		}
		++length;
		--number;
	}
}

int main(int argc, char *argv[])
{

	// Open four files
	ifstream graphname;
	ifstream inputneurons;
	ifstream outputneurons;
	ofstream neuronpaths;

	if(argc != 5)
	{
		cout << "Usage: neuronpaths graphname inputneurons outputneurons neuronpathlist " << endl;
		return 0;
	}
	else
	{
		graphname.open(argv[1]);
		if(!graphname)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
		}
		else
			cout << "Opened graph file " << argv[1] << " successfully." << endl;
		inputneurons.open(argv[2]);
		if(!inputneurons)
		{
			cout << "Could not open input neuron file!" << endl;
			return 0;
		}
		else
```

```
		cout << "Opened input file " << argv[2] << " successfully." << endl;
	outputneurons.open(argv[3]);
	if(!outputneurons)
	{
		cout << "Could not open output neuron file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[3] << " successfully." << endl;
	neuronpaths.open(argv[4]);
	if(!neuronpaths)
	{
		cout << "Could not open neuron path output file!" << endl;
		return 0;
	}
	else
		cout << "Opened output file " << argv[4] << " successfully." << endl;


}



// Load the input neuron vector

int numinputnodes = 0, numoutputnodes = 0, i = 0;
string temp;
inputneurons >> numinputnodes;
string vinputneurons[numinputnodes];
for(i = 0; i < numinputnodes; ++i)
{
	inputneurons >> temp;
	vinputneurons[i] = temp;
}

// load the output neuron vector

outputneurons >> numoutputnodes;
string voutputneurons[numoutputnodes];
for(i = 0; i < numoutputnodes; ++i)
{
	outputneurons >> temp;
	voutputneurons[i] = temp;
}

// Open graph
graph<string> g;
g.clear();

graphname >> g;
```

```
	// calculate and write out paths from input neurons to output neurons

	list<string> path;
	vector<list<string> > vpaths;

	int p=0;
	int q=0;
	for(p = 0; p < numinputnodes; ++p)
		for(q = 0; q < numoutputnodes; ++q)
		{
			cout << "Begining neuron is: " << vinputneurons[p] << endl;
			cout << "Ending neuron is: " << voutputneurons[q] << endl;
			 cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g,
vinputneurons[p])].vtxMapLoc)).first, (*(g.vInfo[find(g, voutputneurons[q])].vtxMapLoc)).first, path) << endl;
			cout << "Path is: ";
			writeList(path, "  ");
			cout << endl;
			vpaths.push_back(path);
		}

	vector<list<string> > vsubpaths;

	// Generate all of the subpaths
	int vpathsize = vpaths.size();
	for(int i=0; i < vpathsize; ++i)
		subpaths(vpaths[i], vsubpaths);

	cout << "Contents of vsubpaths: " << endl;
	// foreach subpath, find all paths that go through that subpath
	int vsubpathsize = vsubpaths.size();
	for(int i = 0; i < vsubpathsize; ++i)
		writeList(vsubpaths[i], " ");

	for(int i = 0; i < vsubpathsize; ++i)
	{
		cout << "Subpath is: ";
		writeList(vsubpaths[i], " ");
		cout << endl;
		cout << "Paths going through this subpath are: " << endl;
		for(int j=0; j < vpathsize; ++j)
			if(sublist(vsubpaths[i], vpaths[j]))
				writeList(vpaths[j], " ");
		cout << endl << endl;
	}

	graphname.close();
	inputneurons.close();
	outputneurons.close();
	neuronpaths.close();
```

```
        return 0;
}
```

15. pathgenes()

```
#include <set>
#include <list>
#include <vector>
#include <fstream>
#include <iostream>
#include "Graph.h"
#include "Utilities.h"
#include <string>

class neuron{
	private:
		string name;
		string lineage;
		string maintype;
		string subtype1;
		string subtype2;
		set<string> genes;

	public:
		neuron()
		{}
		void setName(string x)
		{
			name = x;
		}
		void setLineage(string x)
		{
			lineage = x;
		}
		void setMainType(string x)
		{
			maintype = x;
		}
		void setSubType1(string x)
		{
			subtype1 = x;
		}
		void setSubType2(string x)
		{
			subtype2 = x;
		}
		void addGene(string gene)
		{
			genes.insert(gene);
		}
		string getName()
		{
			return name;
```

```
		}
		string getLineage()
		{
			return lineage;
		}
		string getMainType()
		{
			return maintype;
		}
		string getSubType1()
		{
			return subtype1;
		}
		string getSubType2()
		{
			return subtype2;
		}
		set<string> getGenes()
		{
			return genes;
		}
	};


// SET FUNCTION IMPLEMENTATIONS

template <typename T>
bool operator== (const set<T>& lhs, const set<T>& rhs)
{
	set<T>::const_iterator myself = lhs.begin(), other = rhs.begin();

	// return false if the sets do not have the same size
	if (lhs.size() == rhs.size())
	{
		// compare until encounter end of the sets or
		// find two elements that are not equal
		while (myself != lhs.end() && *myself++ == *other++);

		// if we left the loop before reaching the end
		// of the sets, they are not equal
		if (myself != lhs.end())
			return false;
		else
			return true;
	}
	else
		return false;
}

template <typename T>
set<T> operator+ (const set<T>& lhs, const set<T>& rhs)
{
```

```
	// constuct union
	set<T> setUnion;
	// iterators that traverse the sets
	set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*rhsIter++);
		else
		{
			// the two values are equal. insert just one and
			// move both itertors forward
			setUnion.insert(*lhsIter++);
			rhsIter++;
		}

		// flush any remaining items
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setUnion.insert(*lhsIter++);
		else if (rhsIter != rhs.end())
			while (rhsIter != rhs.end())
				setUnion.insert(*rhsIter++);

	return setUnion;
}

template <typename T>
set<T> operator* (const set<T>& lhs, const set<T>& rhs)
{
	// constuct intersection
	set<T> setIntersection;
	// iterators that traverse the sets
	set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter is in lhs and not in rhs. move iterator
			// forward
			lhsIter++;
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in rhs and not in lhs. move iterator
```

```
                        // forward
                        rhsIter++;
                }
                else
                {
                        // the same value is in both sets. insert one value
                        // and move the iterators forward
                        setIntersection.insert(*lhsIter);
                        lhsIter++;
                        rhsIter++;
                }

                return setIntersection;
}

template <typename T>
set<T> operator- (const set<T>& lhs, const set<T>& rhs)
{
        // construct difference
        set<T> setDifference;
        // iterators that traverse the sets
        set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

        // move forward as long as we have not reached the end of
        // either set
        while (lhsIter != lhs.end() && rhsIter != rhs.end())
                if (*lhsIter < *rhsIter)
                        // *lhsIter belongs to lhs but not to rhs. put it in
                        // the difference
                        setDifference.insert(*lhsIter++);
                else if (*rhsIter < *lhsIter)
                        // *rhsIter is in the rhs but not in the lhs. pass
                        // over it
                        rhsIter++;
                else
                {
                        // the same value is in both sets. move the
                        // iterators forward
                        lhsIter++;
                        rhsIter++;
                }

                // flush any remaining items from lhs
                if (lhsIter != lhs.end())
                        while (lhsIter != lhs.end())
                                setDifference.insert(*lhsIter++);

        return setDifference;
}


template<typename T>
int find(graph<T> &g, string x)
```

```
{
	int r = 0;
	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
			return -1;
		}
}

set<string> findGeneSet(string neuronname, neuron neuron[], int numneurons)
{
	for(int i=0; i < numneurons; ++i)
		if(neuronname == neuron[i].getName())
			return neuron[i].getGenes();
}

int main(int argc, char *argv[])
{
	// cout << "Entered main" << endl;
	ifstream neurons;
	string inputneuron, outputneuron;


	if(argc != 4)
	{
		cout << "Usage: pathgenes graphname inputneuron outputneuron " << endl;
		return 0;
	}
	else
	{
		neurons.open(argv[1]);
		if(!neurons)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
		}
		else
			cout << "Opened graph file " << argv[1] << " successfully." << endl;
		inputneuron = argv[2];
		if(isdigit(inputneuron[0]))
		{
			cout << "Second parameter must be a string: neuron name!" << endl;
			return 0;
		}
		else
			cout << "Second parameter is correct." << endl;
		outputneuron = argv[3];
```

```
	if(isdigit(outputneuron[0]))
	{
		cout << "Third parameter must be a string:  a neuron name" << endl;
		return 0;
	}
	else
		cout << "Third parameter is correct." << endl;
}



int numneurons = 0;
string gene;
vector<neuron> vneurons(numneurons);
neurons >> numneurons;
string temp;
// Create an array of neuron objects
neuron n[numneurons];
for(int i = 0; i < numneurons; ++i)
{
	neurons >> temp;
	n[i].setName(temp);
	neurons >> temp;
	n[i].setLineage(temp);
	neurons >> temp;
	n[i].setMainType(temp);
	neurons >> temp;
	n[i].setSubType1(temp);
	neurons >> temp;
	n[i].setSubType2(temp);
	while(1)
	{
		neurons >> gene;
			if(gene == "end")
				break;
		n[i].addGene(gene);
	}
}
// insert vertices
graph<string> g;

int vneuronsize = vneurons.size();

for(int i=0; i < numneurons; ++i)
	g.insertVertex(n[i].getName());

int numedges = 0;
string neuron1, neuron2;
int weight = 0;

neurons >> numedges;
```

```
	for(int i = 0; i < numedges; ++i)
	{
		neurons >> neuron1;
		neurons >> neuron2;
		neurons >> weight;

		g.insertEdge(neuron1, neuron2, weight);
	}



	list<string> path;

	cout << "Begining neuron is: " << inputneuron << endl;
	cout << "Ending neuron is: " << outputneuron << endl;
	cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g, inputneuron)].vtxMapLoc)).first,
(*(g.vInfo[find(g, outputneuron)].vtxMapLoc)).first, path) << endl;
	cout << "Path is: ";
	writeList(path, "  ");
	cout << endl;

	vector<set<string> > vs;

	list<string>::iterator liter;
	liter = path.begin();
	set<string>::iterator fsiter;
	set<string>::iterator esiter;

	while(liter != path.end())
	{
		vs.push_back(findGeneSet(*liter, n, numneurons));
		cout << endl << endl << "Neuron: " << *liter << " has the following genes: " << endl;
		fsiter = vs.back().begin();
		esiter = vs.back().end();
		writeContainer(fsiter, esiter, " ");
				++liter;
	}


	cout << endl << endl;
	cout << "Genes involved in pathway: ";
	writeList(path, " ");
	cout << "are: " << endl << endl;

	set<string> us;

	us = vs[0];

	for(int i=0; i < vs.size(); ++i)
		us = us + vs[i];
```

```
	fsiter = us.begin();
	esiter = us.end();
	writeContainer(fsiter, esiter, " ");
	if(us.empty())
		cout << "There are no genes involved in this pathway in our database." << endl;

	set<string> is;

	is = vs[0];

	for(int i = 1; i < vs.size(); ++i)
	{
		is = is * vs[i];
	}

	fsiter = is.begin();
	esiter = is.end();
	cout << endl << endl << "The genes that are common to every neuron in this pathway are: " << endl;
	writeContainer(fsiter, esiter, " ");
	if(is.empty())
		cout << "There are no genes common to every neuron in this pathway according to our database."
<< endl;


	return 0;
}
```

16. pathgenespairs

```
#include <set>
#include <list>
#include <vector>
#include <fstream>
#include <iostream>
#include "d_graph.h"
#include "d_util.h"
#include <string>


class neuron{
	private:
		string name;
		string lineage;
		string maintype;
		string subtype1;
		string subtype2;
		set<string> genes;

	public:
		neuron()
		{}
		void setName(string x)
		{
			name = x;
		}
		void setLineage(string x)
		{
			lineage = x;
		}
		void setMainType(string x)
		{
			maintype = x;
		}
		void setSubType1(string x)
		{
			subtype1 = x;
		}
		void setSubType2(string x)
		{
			subtype2 = x;
		}
		void addGene(string gene)
		{
			genes.insert(gene);
		}
		string getName()
		{
```

```
                return name;
        }
        string getLineage()
        {
                return lineage;
        }
        string getMainType()
        {
                return maintype;
        }
        string getSubType1()
        {
                return subtype1;
        }
        string getSubType2()
        {
                return subtype2;
        }
        set<string> getGenes()
        {
                return genes;
        }
};


// SET FUNCTION IMPLEMENTATIONS

template <typename T>
bool operator== (const set<T>& lhs, const set<T>& rhs)
{
        typename set<T>::const_iterator myself = lhs.begin(), other = rhs.begin();

        // return false if the sets do not have the same size
        if (lhs.size() == rhs.size())
        {
                // compare until encounter end of the sets or
                // find two elements that are not equal
                while (myself != lhs.end() && *myself++ == *other++);

                // if we left the loop before reaching the end
                // of the sets, they are not equal
                if (myself != lhs.end())
                        return false;
                else
                        return true;
        }
        else
                return false;
}

template <typename T>
set<T> operator+ (const set<T>& lhs, const set<T>& rhs)
```

```
{
	// constuct union
	set<T> setUnion;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*rhsIter++);
		else
		{
			// the two values are equal. insert just one and
			// move both itertors forward
			setUnion.insert(*lhsIter++);
			rhsIter++;
		}

		// flush any remaining items
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setUnion.insert(*lhsIter++);
		else if (rhsIter != rhs.end())
			while (rhsIter != rhs.end())
				setUnion.insert(*rhsIter++);

	return setUnion;
}

template <typename T>
set<T> operator* (const set<T>& lhs, const set<T>& rhs)
{
	// constuct intersection
	set<T> setIntersection;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter is in lhs and not in rhs. move iterator
			// forward
			lhsIter++;
		else if (*rhsIter < *lhsIter)
```

```
                        // *rhsIter is in rhs and not in lhs. move iterator
                        // forward
                        rhsIter++;
                else
                {
                        // the same value is in both sets. insert one value
                        // and move the iterators forward
                        setIntersection.insert(*lhsIter);
                        lhsIter++;
                        rhsIter++;
                }

                return setIntersection;
}

template <typename T>
set<T> operator- (const set<T>& lhs, const set<T>& rhs)
{
        // construct difference
        set<T> setDifference;
        // iterators that traverse the sets
        typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

        // move forward as long as we have not reached the end of
        // either set
        while (lhsIter != lhs.end() && rhsIter != rhs.end())
                if (*lhsIter < *rhsIter)
                        // *lhsIter belongs to lhs but not to rhs. put it in
                        // the difference
                        setDifference.insert(*lhsIter++);
                else if (*rhsIter < *lhsIter)
                        // *rhsIter is in the rhs but not in the lhs. pass
                        // over it
                        rhsIter++;
                else
                {
                        // the same value is in both sets. move the
                        // iterators forward
                        lhsIter++;
                        rhsIter++;
                }

                // flush any remaining items from lhs
                if (lhsIter != lhs.end())
                        while (lhsIter != lhs.end())
                                setDifference.insert(*lhsIter++);

        return setDifference;
}


template<typename T>
```

```
int find(graph<T> &g, string x)
{
	int r = 0;
	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
			return -1;
		}
}

set<string> findGeneSet(string neuronname, neuron neuron[], int numneurons)
{
	for(int i=0; i < numneurons; ++i)
		if(neuronname == neuron[i].getName())
			return neuron[i].getGenes();
}

vector<string> list2vector(list<string> l)
{
	vector<string> v;
	list<string>::iterator liter;
	liter = l.begin();
	while(liter != l.end())
	{
		v.push_back(*liter++);
	}
	return v;
}


int main(int argc, char *argv[])
{
	ifstream neurons;
	string inputneuron, outputneuron;


	if(argc != 4)
	{
		cout << "Usage: pathgenespairs graphname inputneuron outputneuron " << endl;
		return 0;
	}
	else
	{
		neurons.open(argv[1]);
		if(!neurons)
		{
			cout << "Could not open graph file!" << endl;
```

```
                return 0;
        }
        else
                cout << "Opened graph file " << argv[1] << " successfully." << endl;
        inputneuron = argv[2];
        if(isdigit(inputneuron[0]))
        {
                cout << "Second parameter must be a string: neuron name!" << endl;
                return 0;
        }
        else
                cout << "Second parameter is correct." << endl;
        outputneuron = argv[3];
        if(isdigit(outputneuron[0]))
        {
                cout << "Third parameter must be a string:  a neuron name" << endl;
                return 0;
        }
        else
                cout << "Third parameter is correct." << endl;
}



int numneurons = 0;
string gene;
vector<neuron> vneurons(numneurons);
neurons >> numneurons;
string temp;
neuron n[numneurons];
for(int i = 0; i < numneurons; ++i)
{
        neurons >> temp;
        n[i].setName(temp);
        neurons >> temp;
        n[i].setLineage(temp);
        neurons >> temp;
        n[i].setMainType(temp);
        neurons >> temp;
        n[i].setSubType1(temp);
        neurons >> temp;
        n[i].setSubType2(temp);
        while(1)
        {
                neurons >> gene;
                        if(gene == "end")
                                break;
                n[i].addGene(gene);
        }
}
// insert vertices
graph<string> g;
```

```
	int vneuronsize = vneurons.size();

	for(int i=0; i < numneurons; ++i)
		g.insertVertex(n[i].getName());

	int numedges = 0;
	string neuron1, neuron2;
	int weight = 0;

	neurons >> numedges;

	for(int i = 0; i < numedges; ++i)
	{
		neurons >> neuron1;
		neurons >> neuron2;
		neurons >> weight;

		g.insertEdge(neuron1, neuron2, weight);
	}



	list<string> path;

	cout << "Begining neuron is: " << inputneuron << endl;
	cout << "Ending neuron is: " << outputneuron << endl;
	cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g, inputneuron)].vtxMapLoc)).first,
(*(g.vInfo[find(g, outputneuron)].vtxMapLoc)).first, path) << endl;
	cout << "Path is: ";
	writeList(path, "  ");
	cout << endl;

	vector<set<string> > vs;

	list<string>::iterator liter;
	liter = path.begin();
	set<string>::iterator fsiter;
	set<string>::iterator esiter;

	while(liter != path.end())
	{
		vs.push_back(findGeneSet(*liter, n, numneurons));
		cout << endl << endl << "Neuron: " << *liter << " has the following genes: " << endl;
		fsiter = vs.back().begin();
		esiter = vs.back().end();
		writeContainer(fsiter, esiter, " ");
				++liter;
	}


	cout << endl << endl;
```

```
	cout << "Genes involved in pathway: ";
	writeList(path, " ");
	cout << "are: " << endl << endl;

	set<string> us;
	us = vs[0];
	for(int i=0; i < vs.size(); ++i)
		us = us + vs[i];
	fsiter = us.begin();
	esiter = us.end();
	writeContainer(fsiter, esiter, " ");
	if(us.empty())
		cout << "There are no genes involved in this pathway in our database." << endl;

	vector<string> vpath;

	vpath  = list2vector(path);

	int index = 0;
	int v = 0;

	cout << endl << endl << "Genes common to CONTIGUOUS neurons in the path: " << endl;

	while(v < vpath.size()-1)
	{
		cout << endl << endl << "The genes common to pair " << vpath[v] << " " << vpath[v+1]  << "
are: " << endl;
	set<string> pair;
	pair = vs[index] * vs[index+1];
	if(pair.empty())
		cout << "There are no genes common to these two neurons." << endl;
	else {
		fsiter = pair.begin();
		esiter = pair.end();
		writeContainer(fsiter, esiter, " ");
		}
	++index;
	++v;
	}


	cout << endl << "Genes common to NON-CONTIGUOUS neurons in the pathway:" << endl << endl;

	for(int i = 0; i < vpath.size() - 2; ++i)
		for(int j = 2; j+i < vpath.size(); ++j)
		{
			cout << endl << "The genes common to the non-contiguous pair " << vpath[i] << " " <<
vpath[i+j] << " are: " << endl;
			set<string> pair;
			pair = vs[i] * vs[i+j];
			if(pair.empty())
				cout << "There are no genes common to these two neurons" << endl;
```

```
                        else
                                {
                                        fsiter = pair.begin();
                                        esiter = pair.end();
                                        writeContainer(fsiter, esiter, " ");
                                }
                }


        set<string> is;

        is = vs[0];

        for(int i = 1; i < vs.size(); ++i)
        {
                is = is * vs[i];
        }

        fsiter = is.begin();
        esiter = is.end();
        cout << endl << endl << "The genes that are common to every neuron in this pathway are: " << endl;
        writeContainer(fsiter, esiter, " ");
        if(is.empty())
                cout << "There are no genes common to every neuron in this pathway according to our database."
<< endl;


        return 0;
}
```

17. develstages()

```
#include <set>
#include <list>
#include <vector>
#include <fstream>
#include <iostream>
#include "d_util.h"
#include <string>


class neuron{
	private:
		string name;
		string lineage;
		string maintype;
		string subtype1;
		string subtype2;
		set<string> genes;

	public:
		neuron()
		{}
		void setName(string x)
		{
			name = x;
		}
		void setLineage(string x)
		{
			lineage = x;
		}
		void setMainType(string x)
		{
			maintype = x;
		}
		void setSubType1(string x)
		{
			subtype1 = x;
		}
		void setSubType2(string x)
		{
			subtype2 = x;
		}
		void addGene(string gene)
		{
			genes.insert(gene);
		}
		string getName()
```

```
		{
			return name;
		}
		string getLineage()
		{
			return lineage;
		}
		string getMainType()
		{
			return maintype;
		}
		string getSubType1()
		{
			return subtype1;
		}
		string getSubType2()
		{
			return subtype2;
		}
		set<string> getGenes()
		{
			return genes;
		}
	};

class syn {
		public:
			string n1;
			string n2;
			int wt;
		};

// SET FUNCTION IMPLEMENTATIONS

template <typename T>
bool operator== (const set<T>& lhs, const set<T>& rhs)
{
	typename set<T>::const_iterator myself = lhs.begin(), other = rhs.begin();

	// return false if the sets do not have the same size
	if (lhs.size() == rhs.size())
	{
		// compare until encounter end of the sets or
		// find two elements that are not equal
		while (myself != lhs.end() && *myself++ == *other++);

		// if we left the loop before reaching the end
		// of the sets, they are not equal
		if (myself != lhs.end())
			return false;
		else
			return true;
```

```
	}
	else
		return false;
}

template <typename T>
set<T> operator+ (const set<T>& lhs, const set<T>& rhs)
{
	// constuct union
	set<T> setUnion;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter belongs to the union. insert and
			// move iterator forward
			setUnion.insert(*rhsIter++);
		else
		{
			// the two values are equal. insert just one and
			// move both itertors forward
			setUnion.insert(*lhsIter++);
			rhsIter++;
		}

		// flush any remaining items
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
				setUnion.insert(*lhsIter++);
		else if (rhsIter != rhs.end())
			while (rhsIter != rhs.end())
				setUnion.insert(*rhsIter++);

	return setUnion;
}

template <typename T>
set<T> operator* (const set<T>& lhs, const set<T>& rhs)
{
	// constuct intersection
	set<T> setIntersection;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
```

```
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter is in lhs and not in rhs. move iterator
			// forward
			lhsIter++;
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in rhs and not in lhs. move iterator
			// forward
			rhsIter++;
		else
		{
			// the same value is in both sets. insert one value
			// and move the iterators forward
			setIntersection.insert(*lhsIter);
			lhsIter++;
			rhsIter++;
		}

		return setIntersection;
}

template <typename T>
set<T> operator- (const set<T>& lhs, const set<T>& rhs)
{
	// construct difference
	set<T> setDifference;
	// iterators that traverse the sets
	typename set<T>::const_iterator lhsIter = lhs.begin(), rhsIter = rhs.begin();

	// move forward as long as we have not reached the end of
	// either set
	while (lhsIter != lhs.end() && rhsIter != rhs.end())
		if (*lhsIter < *rhsIter)
			// *lhsIter belongs to lhs but not to rhs. put it in
			// the difference
			setDifference.insert(*lhsIter++);
		else if (*rhsIter < *lhsIter)
			// *rhsIter is in the rhs but not in the lhs. pass
			// over it
			rhsIter++;
		else
		{
			// the same value is in both sets. move the
			// iterators forward
			lhsIter++;
			rhsIter++;
		}

		// flush any remaining items from lhs
		if (lhsIter != lhs.end())
			while (lhsIter != lhs.end())
```

```
                                        setDifference.insert(*lhsIter++);

        return setDifference;
}

set<string> findGeneSet(string neuronname, neuron neuron[], int numneurons)
{
        for(int i=0; i < numneurons; ++i)
                if(neuronname == neuron[i].getName())
                        return neuron[i].getGenes();
}

int min(neuron n[], int numneurons)
{
        int min = 100;
        for(int i = 0; i < numneurons; ++i)
                if(n[i].getLineage().length() < min)
                        min = n[i].getLineage().length();
        return min;
}

int max(neuron n[], int numneurons)
{
        int max = 0;
        for(int i = 0; i < numneurons; ++i)
                if(n[i].getLineage().length() > max)
                        max = n[i].getLineage().length();
        return max;
}

bool find(vector<string> n, string x)
{
        int vsize = n.size();
        for(int i = 0; i < vsize; ++i)
                if(n[i] == x)
                        return true;
                else if(i == vsize-1)
                        return false;
}


template< typename Iterator>
void writeContainerFile(Iterator first, Iterator last, const string& seperator, ofstream& output)
{
        Iterator iter = first;
        while(iter != last)
        {
                output << *iter << seperator;
                iter++;
        }
}
```

```
int main(int argc, char *argv[])
{
	ifstream neurons;
	string inputneuron, outputneuron;
	ofstream outfile;

	if(argc != 3)
	{
		cout << "Usage: develstages graphname outputfile " << endl;
		return 0;
	}
	else
	{
		neurons.open(argv[1]);
		if(!neurons)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
		}
		else
			cout << "Opened graph file " << argv[1] << " successfully." << endl;
		outfile.open(argv[2]);
		if(!outfile)
		{
			cout << "Could not open output file!" << endl;
			return 0;
		}
		else
			cout << "Opened output file " << argv[2] << " successfully." << endl;
	}



	int numneurons = 0;
	string gene;
	vector<neuron> vneurons(numneurons);
	neurons >> numneurons;
	string temp;
	neuron n[numneurons];
	int counter = 0;
	for(int i = 0; i < numneurons; ++i)
	{
		neurons >> temp;
		n[i].setName(temp);
		++counter;
		neurons >> temp;
		n[i].setLineage(temp);
		++counter;
		neurons >> temp;
		n[i].setMainType(temp);
		++counter;
```

```
                neurons >> temp;
                n[i].setSubType1(temp);
                ++counter;
                neurons >> temp;
                n[i].setSubType2(temp);
                ++counter;
                while(1)
                {
                        neurons >> gene;
                                if(gene == "end")
                                        break;
                        n[i].addGene(gene);
                        ++counter;
                }
        }

int maxstring = max(n, numneurons);
int minstring = min(n, numneurons);
int numfiles = maxstring - minstring + 1;

int filenum = 0;

int stage = 1;

set<string> already_genes;

for(int length=minstring; length <= maxstring; ++length)
{
        vector<string> vneur;
        vector<string> vnewneur;
        vector<string> vs;
        vector<string> vm;
        vector<string> vi;
        vector<string> vcs;
        vector<string> vcm;
        vector<string> vci;

        int stageedges = 0;
        outfile << endl << endl << "Development Stage: " << stage << endl << endl;

        // Find all neurons of lineage string length length or less
        int i = 0;
        for(i = 0; i < numneurons; ++i)
                if(n[i].getLineage().length() <= length)
                {
                        vneur.push_back(n[i].getName());
                        if(n[i].getMainType() == "SN")
                                vs.push_back(n[i].getName());
                        else if(n[i].getMainType() == "MN")
                                vm.push_back(n[i].getName());
                        else if(n[i].getMainType() == "IN")
                                vi.push_back(n[i].getName());
```

```
		}


if(stage > 1)
{

for(int i = 0; i < numneurons; ++i)
{
		if(n[i].getLineage().length() == length)
			vnewneur.push_back(n[i].getName());
		if((n[i].getLineage().length() == length) && (n[i].getMainType() == "SN"))
			vcs.push_back(n[i].getName());
		else if((n[i].getLineage().length() == length) && (n[i].getMainType() == "MN"))
			vcm.push_back(n[i].getName());
		else if((n[i].getLineage().length() == length) && (n[i].getMainType() == "IN"))
			vci.push_back(n[i].getName());
}
}

if(!vnewneur.empty())
{
		outfile << endl << "New neurons at this stage: " << endl;
		writeContainerFile(vnewneur.begin(), vnewneur.end(), " ", outfile);
		outfile << endl << "New sensory neurons at this stage: " << endl;
		writeContainerFile(vcs.begin(), vcs.end(), " ", outfile);

		outfile << endl << "New motor neurons at this stage: " << endl;
		writeContainerFile(vcm.begin(), vcm.end(), " ", outfile);

		outfile << endl << "New inter-neurons at this stage: " << endl;
		writeContainerFile(vci.begin(), vci.end(), " ", outfile);
}
		outfile << endl << "All Sensory neurons at this stage: " << endl;
		writeContainerFile(vs.begin(), vs.end(), " ", outfile);

		outfile << endl << "All Motor neurons at this stage: " << endl;
		writeContainerFile(vm.begin(), vm.end(), " ", outfile);

		outfile << endl << "All Inter-neurons at this stage: " << endl;
		writeContainerFile(vi.begin(), vi.end(), " ", outfile);

outfile << endl << "Genes at this development stage: " << endl;
vector<set<string> > vs_stage;
vector<set<string> > vs_sensory;
vector<set<string> > vs_motor;
vector<set<string> > vs_inter;
vector<set<string> > vs_new_sensory;
vector<set<string> > vs_new_motor;
vector<set<string> > vs_new_inter;
set<string> newly_activated_genes;

int vneursize = vneur.size();
```

```
for(int i = 0; i < vneursize; ++i)
	vs_stage.push_back(findGeneSet(vneur[i], n, numneurons));
int vssize = vs.size();
for(int i = 0; i < vssize; ++i)
	vs_sensory.push_back(findGeneSet(vs[i], n, numneurons));
int vmsize = vm.size();
for(int i = 0; i < vmsize; ++i)
	vs_motor.push_back(findGeneSet(vm[i], n, numneurons));
int visize = vi.size();
for(int i = 0; i < visize; ++i)
	vs_inter.push_back(findGeneSet(vi[i], n, numneurons));
set<string>::iterator fsiter;
set<string>::iterator esiter;
set<string> us_stage;
us_stage = vs_stage[0];
for(int i = 0; i < vs_stage.size(); ++i)
	us_stage = us_stage + vs_stage[i];
if(stage == 1)
	already_genes = us_stage;
fsiter = us_stage.begin();
esiter = us_stage.end();
writeContainerFile(fsiter, esiter, " ", outfile);
if(us_stage.empty())
	cout << "There are no genes at this development stage." << endl;

set<string> us_sensory;
set<string> us_motor;
set<string> us_inter;

if(!vs_sensory.empty())
{
us_sensory = vs_sensory[0];
for(int i = 0; i < vs_sensory.size(); ++i)
	us_sensory = us_sensory + vs_sensory[i];
outfile << endl << "Sensory Neuron Genes at this stage: " << endl;
fsiter = us_sensory.begin();
esiter = us_sensory.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
	outfile << "There are no sensory neuron genes at this stage." << endl;

if(!vs_motor.empty())
{
us_motor = vs_motor[0];
for(int i = 0; i < vs_motor.size(); ++i)
	us_motor = us_motor + vs_motor[i];
outfile << endl << "Motor Neuron Genes at this stage: " << endl;
fsiter = us_motor.begin();
esiter = us_motor.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
```

```
else
	outfile << "There are no motor neuron genes at this stage." << endl;

if(!vs_inter.empty())
{
us_inter = vs_inter[0];
for(int i = 0; i < vs_inter.size(); ++i)
	us_inter = us_inter + vs_inter[i];
fsiter = us_inter.begin();
esiter = us_inter.end();
outfile << endl << "Inter-neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
	outfile << "There are no inter-neuron genes at this stage." << endl;

outfile << endl << "Sensory-specific genes: " << endl;
set<string> ss;
ss = us_sensory - us_motor - us_inter;
if(!ss.empty())
{
fsiter = ss.begin();
esiter = ss.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
	outfile << endl << "There are no sensory-specific genes." << endl;

outfile << endl << "Motor-specific genes: " << endl;
set<string> ms;
ms = us_motor - us_sensory - us_inter;
if(!ms.empty())
{
fsiter = ms.begin();
esiter = ms.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
	outfile << endl << "There are no motor-specific genes." << endl;

outfile << endl << "Inter-neuron specific genes:" << endl;
set<string> is;
is = us_inter - us_sensory - us_motor;
if(!is.empty())
{
fsiter = is.begin();
esiter = is.end();
writeContainerFile(fsiter, esiter, " ", outfile);
}
else
	outfile << endl << "There are no interneuron specific genes." << endl;
```

```
	outfile << endl << "Genes common to sensory neurons, motor neurons, and interneurons at this stage: " <<
endl;
	set<string> common;
	common = us_inter * us_sensory * us_motor;
	if(!common.empty())
	{
	fsiter = common.begin();
	esiter = common.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
	}
	else
		outfile << endl << "There are no genes common to sensory neurons, motor neurons, and
interneurons at this stage." << endl;


	if(stage > 1)
	{

	if(!vnewneur.empty())
	{
	outfile << endl << "New genes at this stage: " << endl;
    int vnewneursize = vnewneur.size();
	vector<set<string> > vs_new_stage;
	for(int i = 0; i < vnewneursize; ++i)
		vs_new_stage.push_back(findGeneSet(vnewneur[i], n, numneurons));

	set<string> us_new_stage;
	us_new_stage = vs_new_stage[0];
	for(int i = 0; i < vs_new_stage.size(); ++i)
		us_new_stage = us_new_stage + vs_new_stage[i];
	newly_activated_genes = us_new_stage - already_genes;
	fsiter = us_new_stage.begin();
	esiter = us_new_stage.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
	if(us_new_stage.empty())
		outfile << "There are no new genes at this development stage." << endl;

	if(!newly_activated_genes.empty())
	{
		outfile << endl << "Newly activated genes this stage: " << endl;
		fsiter = newly_activated_genes.begin();
		esiter = newly_activated_genes.end();
		writeContainerFile(fsiter, esiter, " ", outfile);
	}
	else
		outfile << endl << "There are no newly activated genes this stage." << endl;



	set<string> us_new_sensory;
	set<string> us_new_motor;
	set<string> us_new_inter;
```

```
if(!vcs.empty())
{
int vcssize = vcs.size();
for(int i = 0; i < vcssize; ++i)
	vs_new_sensory.push_back(findGeneSet(vcs[i], n, numneurons));
us_new_sensory = vs_new_sensory[0];
for(int i = 0; i < vs_new_sensory.size(); ++i)
	us_new_sensory = us_new_sensory + vs_new_sensory[i];
fsiter = us_new_sensory.begin();
esiter = us_new_sensory.end();
outfile << endl << "New sensory neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}

if(!vcm.empty())
{
int vcmsize = vcm.size();
for(int i = 0; i < vcmsize; ++i)
	vs_new_motor.push_back(findGeneSet(vcm[i], n, numneurons));
us_new_motor = vs_new_motor[0];
for(int i = 0; i < vs_new_motor.size(); ++i)
	us_new_motor = us_new_motor + vs_new_motor[i];
fsiter = us_new_motor.begin();
esiter = us_new_motor.end();
outfile << endl << "New motor neuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}

if(!vci.empty())
{
int vcisize = vci.size();
for(int i = 0; i < vcisize; ++i)
	vs_new_inter.push_back(findGeneSet(vci[i], n, numneurons));
us_new_inter = vs_new_inter[0];
for(int i = 0; i < vs_new_inter.size(); ++i)
	us_new_inter = us_new_inter + vs_new_inter[i];
fsiter = us_new_inter.begin();
esiter = us_new_inter.end();
outfile << endl << "New interneuron genes at this stage: " << endl;
writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "New sensory-specific genes: " << endl;
set<string> us_new_spec_sensory;
us_new_spec_sensory = us_new_sensory - us_new_motor - us_new_inter;
if(!us_new_spec_sensory.empty())
{
	fsiter = us_new_spec_sensory.begin();
	esiter = us_new_spec_sensory.end();
	writeContainerFile(fsiter, esiter, " ", outfile);
}
```

```
outfile << endl << "New motor-specific genes: " << endl;
set<string> us_new_spec_motor;
us_new_spec_motor = us_new_motor - us_new_sensory - us_new_inter;
if(!us_new_spec_motor.empty())
{
        fsiter = us_new_spec_motor.begin();
        esiter = us_new_spec_motor.end();
        writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "New interneuron-specific genes: " << endl;
set<string> us_new_spec_inter;
us_new_spec_inter = us_new_inter - us_new_sensory - us_new_motor;
if(!us_new_spec_inter.empty())
{
        fsiter = us_new_spec_inter.begin();
        esiter = us_new_spec_inter.end();
        writeContainerFile(fsiter, esiter, " ", outfile);
}

outfile << endl << "Genes common to new neurons at this stage: " << endl;
set<string> new_common;
new_common = us_new_sensory * us_new_motor * us_new_inter;
if(!new_common.empty());
{
        fsiter = new_common.begin();
        esiter = new_common.end();
        writeContainerFile(fsiter, esiter, " ", outfile);
}

already_genes = already_genes + us_new_stage;

}
}

// write out neuron list
vneursize = vneur.size();
outfile << endl << vneursize << endl;
for(int i=0; i < vneursize; ++i)
        outfile << vneur[i] << endl;


vector<syn> postsyn;
vector<syn> presyn;
vector<syn> newconnection;
vector<syn> oldconnection;
syn vsyn;

// rewind file here
neurons.clear();
neurons.seekg(0);
```

```
	string temp;
	for(int i = 0; i < counter+numneurons+1; ++i)
		neurons >> temp;
	int numedges = 0;
	string neuron1, neuron2;
	int weight = 1;
	neurons >> numedges;

	for(int i = 0; i < numedges; ++i)
	{
		neurons >> neuron1;
		neurons >> neuron2;
		neurons >> weight;
		if(find(vneur, neuron1) && find(vneur, neuron2))
		{
			outfile << neuron1 << " " << neuron2 << " " << weight << endl;
			++stageedges;
		}

		if(stage > 1)
		{
		if((find(vnewneur, neuron1) == true) && (find(vnewneur, neuron2) == false) && (find(vneur,
neuron2) == true))
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			presyn.push_back(vsyn);
		}
		else if ((find(vnewneur, neuron1) == false) && (find(vneur, neuron1) == true) &&
(find(vnewneur, neuron2) == true))
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			postsyn.push_back(vsyn);
		}
		else if ((find(vnewneur, neuron1) == true) && (find(vnewneur, neuron2) == true))
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			newconnection.push_back(vsyn);
		}
		else if ((find(vnewneur, neuron1) == false) && (find(vnewneur, neuron2) == false))
		{
			vsyn.n1 = neuron1;
			vsyn.n2 = neuron2;
			vsyn.wt = weight;
			oldconnection.push_back(vsyn);
		}
		}
```

```
		}
		outfile << stageedges;

		if(stage > 1)
		{
		outfile << endl << endl << "Nature of new synapses: " << endl << endl;
		outfile << "Pre-synaptic: " << presyn.size() << endl;
			for(int i = 0; i < presyn.size(); ++i)
				outfile << presyn[i].n1 << " " << presyn[i].n2 << " " << presyn[i].wt << ", ";
		outfile << endl << "Post-synaptic: " << postsyn.size() << endl;
			for(int i = 0; i < postsyn.size(); ++i)
				outfile << postsyn[i].n1 << " " << postsyn[i].n2 << " " << postsyn[i].wt << ", ";
		outfile << endl << "New connections: " << newconnection.size() << endl;
			for(int i = 0; i < newconnection.size(); ++i)
				outfile << newconnection[i].n1 << " " << newconnection[i].n2 << " " <<
newconnection[i].wt << ", ";
		outfile << endl << endl;
		}
		++stage;
}

outfile.close();
neurons.close();

		return 0;
}
```

Appendix IV

Data Files

1. lineage280.txt
2. Syn2.txt
3. Edges.txt
4. LeftNeurons.txt
5. RightNeurons.txt
6. LeftCorrNeurons.txt
7. RightCorrNeurons.txt
8. LeftLineage
9. RightLineage
10. LeftEdges
11. RightEdges
12. LeftCorrLineage
13. RightCorrLineage
14. LeftCorrEdges
15. RightCorrEdges
16. StrongComp
17. LeftStrongComp
18. RightStrongComp
19. Sensory.txt
20. Motor.txt
21. NeuronPaths

1. Lineage280.txt

The lineage string file for the entire organism *C. elegans*.

```
ADAL            AB.plapaaaapp
ADAR            AB.prapaaaapp
ADEL            AB.plapaaaapa
ADER            AB.prapaaaapa
ADFL            AB.alpppppaa
ADFR            AB.praaappaa
ADLL            AB.alpppaad
ADLR            AB.praaapaad
AFDL            AB.alpppapav
AFDR            AB.praaaapav
AIAL            AB.plppaappa
AIAR            AB.prppaappa
AIBL            AB.plaapappa
AIBR            AB.praapappa
AIML            AB.plpaapppa
AIMR            AB.prpaapppa
AINL            AB.alaaaalal
AINR            AB.alaapaaar
AIYL            AB.plpapaaap
AIYR            AB.prpapaaap
AIZL            AB.plapaaapav
AIZR            AB.prapaaapav
ALA             AB.alapppaaa
ALML            AB.arppaappa
ALMR            AB.arpppappa
ALNL            AB.plapappppap
ALNR            AB.prapappppap
AQR             ABprapapaaQRap
AS1             ABplapaappP1apa
AS2             ABplapaappP2apa
AS3             ABplappaaaP3apa
AS4             ABplappaapP4apa
AS5             ABplappaapP5apa
AS6             ABplappappP6apa
AS7             ABplappappP7apa
AS8             ABplapapapP8apa
AS9             ABplapapapP9apa
AS10            ABplapappa10apa
AS11            ABplapappa11apa
ASEL            AB.alpppppppaa
ASER            AB.praaapppaa
ASGL            AB.plaapapap
ASGR            AB.praapapap
ASHL            AB.plpaappaa
ASHR            AB.prpaappaa
ASIL            AB.plaapapppa
ASIR            AB.praapapppa
ASJL            AB.alpppppppa
```

```
ASJR            AB.praaappppa
ASKL            AB.alpppapppa
ASKR            AB.praaaapppa
AUAL            AB.alpppppppp
AUAR            AB.praaappppp
AVAL            AB.alppaaapa
AVAR            AB.alaappapa
AVBL            AB.plpaapaap
AVBR            AB.prpaapaap
AVDL            AB.alaaapalr
AVDR            AB.alaaapprl
AVEL            AB.alpppaaaa
AVER            AB.praaaaaaa
AVFL            AB.prappaapWaaa
AVFR            ABplapaappP1aaa
AVG             AB.prpapppap
AVHL            AB.alapaaaaa
AVHR            AB.alappapaa
AVJL            AB.alapapppa
AVJR            AB.alapppppa
AVKL            AB.plpapapap
AVKR            AB.prpapapap
AVL             AB.prpappaap
AVM             ABprapapaaQRpaa
AWAL            AB.plaapapaa
AWAR            AB.praapapaa
AWBL            AB.alpppppap
AWBR            AB.praaappap
AWCL            AB.plpaaaaap
AWCR            AB.prpaaaaap
BAGL            AB.alppappap
BAGR            AB.arappppap
BDUL            AB.arppaappp
BDUR            AB.arpppappp
CEPDL           AB.plaaaaappa
CEPDR           AB.arpapaappa
CEPVL           AB.plpaappppa
CEPVR           AB.prpaappppa
DA1             AB.prppapaap
DA2             AB.plppapapa
DA3             AB.prppapapa
DA4             AB.plppapapp
DA5             AB.prppapapp
DA6             AB.plpppaaap
DA7             AB.prpppaaap
DA8             AB.prpapappp
DA9             AB.plpppaaaa
DB1             AB.plpaaaapp
DB2             AB.arappappa
DB3             AB.prpaaaapp
DB4             AB.prpappapp
DB5             AB.plpapappp
DB6             AB.plppaappp
```

DB7	AB.prppaappp
DD1	AB.plppappap
DD2	AB.prppappap
DD3	AB.plppapppa
DD4	AB.prppapppa
DD5	AB.plppapppp
DD6	AB.prppapppp
DVA	AB.prpppppapp
DVB	AB.plpapppaKp
DVC	ABplapaappP2Caapaa
FLPL	AB.plapaaapad
FLPR	AB.prapaaapad
HSNL	AB.plapppappa
HSNR	AB.prapppappa
IL1DL	AB.alapappaaa
IL1DR	AB.alappppaaa
IL1L	AB.alapaappaa
IL1R	AB.alaappppaa
IL1VL	AB.alppapppaa
IL1VR	AB.arappppaa
IL2DL	AB.alapappap
IL2DR	AB.alappppap
IL2L	AB.alapaappp
IL2R	AB.alaappppp
IL2VL	AB.alppappppp
IL2VR	AB.arappppppp
LUAL	AB.plpppaapap
LUAR	AB.prpppaapap
OLLL	AB.alpppapaa
OLLR	AB.praaapapaa
OLQDL	AB.alapapapaa
OLQDR	AB.alapppapaa
OLQVL	AB.plpaaappaa
OLQVR	AB.prpaaappaa
PDA	AB.prpppaaaa
PDB	ABplapappa12apa
PDEL	ABplapapaaVLpaaa
PDER	ABprapapaaVRpaaa
PHAL	AB.plpppaapp
PHAR	AB.prpppaapp
PHBL	AB.plapppappp
PHBR	AB.prapppappp
PHCL	ABplappppTLpppaa
PHCR	ABprappppTRpppaa
PLML	AB.plapappppaa
PLMR	AB.prapappppaa
PLNL	ABplappppTLpppap
PLNR	ABprappppTRpppap
PQR	ABplapapaaQLap
PVCL	AB.plpppaapaa
PVCR	AB.prpppaapaa
PVDL	ABplapapaaVLpaapa
PVDR	ABprapapaaTRpaapa

PVM	ABplapapaaTLpaa
PVNL	ABplappppTLappp
PVNR	ABprappppTRappp
PVPL	AB.plpppaaa
PVPR	AB.prpppaaa
PVQL	AB.plapppaaa
PVQR	AB.prapppaaa
PVR	ABplapaappP2Caappa
PVT	AB.plpappppa
PVWL	ABplappppTLppa
PVWR	ABprappppTRppa
RIAL	AB.alapaapaa
RIAR	AB.alaapppaa
RIBL	AB.plpaappap
RIBR	AB.prpaappap
RICL	AB.plppaaaapp
RICR	AB.prppaaaapp
RID	AB.alappaapa
RIFL	AB.plppapaaap
RIFR	AB.prppapaaap
RIGL	AB.plppappaa
RIGR	AB.prppappaa
RIH	AB.prpappaaa
RIML	AB.plppaapap
RIMR	AB.prppaapap
RIPL	AB.alpapaaaa
RIPR	AB.prappaaaa
RIR	AB.prpapppaa
RIS	AB.prpappapa
RIVL	AB.plpaapaaa
RIVR	AB.prpaapaaa
RMDDL	AB.alpapapaa
RMDDR	AB.arappapaa
RMDL	AB.alpppapad
RMDR	AB.praaaapad
RMDVL	AB.alppapaaa
RMDVR	AB.arapppaaa
RMED	AB.alapppaap
RMEL	AB.alaaaarlp
RMER	AB.alaaaarrp
RMEV	AB.plpappaaa
RMFL	ABplapaapG2al
RMFR	ABplapaapG2ar
RMGL	AB.plapaaapp
RMGR	AB.prapaaapp
RMHL	ABprpaaaapG1l
RMHR	ABprpaaaapG1r
SAADL	AB.alppapapa
SAADR	AB.arapppapa
SAAVL	AB.plpaaaaaa
SAAVR	AB.prpaaaaaa
SABD	AB.plppapaap
SABVL	AB.plppapaaaa

SABVR	AB.prppapaaaa
SDQL	ABplapapaaQLpap
SDQR	ABprapapaaQRpap
SIADL	AB.plpapaapa
SIADR	AB.prpapaapa
SIAVL	AB.plpapappa
SIAVR	AB.prpapappa
SIBDL	AB.plppaaaaa
SIBDR	AB.prppaaaaa
SIBVL	AB.plpapaapp
SIBVR	AB.prpapaapp
SMBDL	AB.alpapapapp
SMBDR	AB.arappapapp
SMBVL	AB.alpapappp
SMBVR	AB.arappappp
SMDDL	AB.plpapaaaa
SMDDR	AB.prpapaaaa
SMDVL	AB.alppappaa
SMDVR	AB.arappppaa
URADL	AB.plaaaaaaa
URADR	AB.arpapaaaa
URAVL	AB.plpaaapaa
URAVR	AB.prpaaapaa
URBL	AB.plaapaapa
URBR	AB.praapaapa
URXL	AB.plaaaaappp
URXR	AB.arpapaappp
URYDL	AB.alapapapp
URYDR	AB.alapppapp
URYVL	AB.plpaaappp
URYVR	AB.prpaaappp
VA1	AB.prapaapWpa
VA2	ABplapaappP2aaaa
VA3	ABplappaaaP3aaaa
VA4	ABplappaaaP4aaaa
VA5	ABplappaapP5aaaa
VA6	ABplappappP6aaaa
VA7	ABplappappP7aaaa
VA8	ABplapapapP8aaaa
VA9	ABplapapapP9aaaa
VA10	ABplapapap10aaaa
VA11	ABplapappa11aaaa
VA12	ABplapappa12aaaa
VB1	ABplapaappP1aaap
VB2	AB.prapaappWapp
VB3	ABplapaappP2aaap
VB4	ABplappaaaP3aaap
VB5	ABplappaaaP4aaap
VB6	ABplappaapP5aaap
VB7	ABplappaapP6aaap
VB8	ABplappappP7aaap
VB9	ABplappappP8aaap
VB10	ABplapapapP9aaap

VB11	ABplapapap10aaap
VC1	ABplappaaaP3aap
VC2	ABplappaaaP4aap
VC3	ABplappaapP5aap
VC4	ABplappaapP6aap
VC5	ABplappappP7aap
VC6	ABplappappP8aap
VD1	AB.prapaapWpp
VD2	ABplapaappP1app
VD3	ABplapaappP2app
VD4	ABplappaaaP3app
VD5	ABplappaaaP4app
VD6	ABplappaapP5app
VD7	ABplappaapP6app
VD8	ABplappappP7app
VD9	ABplappappP8app
VD10	ABplapapapP9app
VD11	ABplapapap10app
VD12	ABplapappa11app
VD13	ABplapappa12app

2. Syn2.txt
This is the Synaptic Graph File for the Entire Organism.
Here represented is only part of the file. To see the entire file, go to the CD.

```
280
ADAL
ADAR
ADEL
ADER
ADFL
ADFR
ADLL
ADLR
AFDL
AFDR
AIAL
AIAR
...
VD5
VD6
VD7
VD8
VD9
VD10
VD11
VD12
VD13
2199
ADAL  AIBL  1
ADAL  AIBR  1
ADAL  AVAR  1
ADAL  AVBL  1
ADAL  AVBR  1
ADAL  AVDL  1
ADAL  AVEL  1
ADAL  AVJR  1
ADAL  FLPR  1
ADAL  RICL  1
ADAL  RICR  1
ADAL  RIML  1
...
AVL   VD12  1
PVNR  VD12  1
VA11  VD12  1
VA12  VD12  1
VB10  VD12  1
AS11  VD13  1
AVAR  VD13  1
DA9   VD13  1
DB7   VD13  1
PDA   VD13  1
```

```
PDB   VD13  1
RID   VD13  1
VA12  VD13  1
```

3. Edges.txt

This file represents all synapses in the mature worm, minus gap junctions and neuromuscular junctions. This file is the same as "Syn2.txt" minus the nodes (neurons) at the beginning. The complete file may be found on the CD.

```
4379
ADAL  AIBL  1
ADAL  AIBR  1
ADAL  AVAR  1
ADAL  AVBL  1
ADAL  AVBR  1
ADAL  AVDL  1
ADAL  AVEL  1
ADAL  AVJR  1
ADAL  FLPR  1
ADAL  RICL  1
ADAL  RICR  1
ADAL  RIML  1
ADAL  RIPL  1
ADAL  SMDVR 1
ADAR  AIBL  1
. . .
AVFR  VD11  1
PDEL  VD11  1
VB10  VD11  1
AVL   VD12  1
PVNR  VD12  1
VA11  VD12  1
VA12  VD12  1
VB10  VD12  1
AS11  VD13  1
AVAR  VD13  1
DA9   VD13  1
DB7   VD13  1
PDA   VD13  1
PDB   VD13  1
RID   VD13  1
VA12  VD13  1
```

4. LeftNeurons.txt.

This is a list of neurons on the left side of the *C. elegans* nervous system.

```
93
ADAL
ADEL
ADFL
ADLL
AFDL
AIAL
AIBL
AIML
AINL
AIYL
AIZL
ALML
ALNL
ASEL
ASGL
ASHL
ASIL
ASJL
ASKL
AUAL
AVAL
AVBL
AVDL
AVEL
AVFL
AVHL
AVJL
AVKL
AVL
AWAL
AWBL
AWCL
BAGL
BDUL
CEPDL
CEPVL
FLPL
HSNL
IL1DL
IL1L
IL1VL
IL2DL
IL2L
IL2VL
LUAL
OLLL
```

OLQDL
OLQVL
PDEL
PHAL
PHBL
PHCL
PLML
PLNL
PVCL
PVDL
PVNL
PVPL
PVQL
PVWL
RIAL
RIBL
RICL
RIFL
RIGL
RIML
RIPL
RIVL
RMDDL
RMDL
RMDVL
RMEL
RMFL
RMGL
RMHL
SAADL
SAAVL
SABVL
SDQL
SIADL
SIAVL
SIBDL
SIBVL
SMBDL
SMBVL
SMDDL
SMDVL
URADL
URAVL
URBL
URXL
URYDL
URYVL

5. Rightneurons.txt.

This is a list of neurons occurring on the right side of the nervous system.

```
96
ADAR
ADER
ADFR
ADLR
AFDR
AIAR
AIBR
AIMR
AINR
AIYR
AIZR
ALMR
ALNR
AQR
ASER
ASGR
ASHR
ASIR
ASJR
ASKR
AUAR
AVAR
AVBR
AVDR
AVER
AVFR
AVHR
AVJR
AVKR
AWAR
AWBR
AWCR
BAGR
BDUR
CEPDR
CEPVR
FLPR
HSNR
IL1DR
IL1R
IL1VR
IL2DR
IL2R
IL2VR
LUAR
OLLR
OLQDR
```

OLQVR
PDER
PHAR
PHBR
PHCR
PLMR
PLNR
PQR
PVCR
PVDR
PVNR
PVPR
PVQR
PVR
PVWR
RIAR
RIBR
RICR
RIFR
RIGR
RIMR
RIPR
RIR
RIVR
RMDDR
RMDR
RMDVR
RMER
RMFR
RMGR
RMHR
SAADR
SAAVR
SABVR
SDQR
SIADR
SIAVR
SIBDR
SIBVR
SMBDR
SMBVR
SMDDR
SMDVR
URADR
URAVR
URBR
URXR
URYDR
URYVR

Place Left and Right Corresponding List Here.

8. LeftLineage. This file contains the lineage strings of just the left side of the nervous system.

```
ADAL AB.plapaaaapp
ADEL AB.plapaaaapa
ADFL AB.alpppppaa
ADLL AB.alpppaad
AFDL AB.alpppapav
AIAL AB.plppaappa
AIBL AB.plaapappa
AIML AB.plpaapppa
AINL AB.alaaaalal
AIYL AB.plpapaaap
AIZL AB.plapaaapav
ALML AB.arppaappa
ALNL AB.plapapppap
ASEL AB.alpppppaa
ASGL AB.plaapapap
ASHL AB.plpaappaa
ASIL AB.plaapapppa
ASJL AB.alpppppppa
ASKL AB.alpppapppa
AUAL AB.alpppppppp
AVAL AB.alppaaapa
AVBL AB.plpaapaap
AVDL AB.alaaapalr
AVEL AB.alpppaaaa
AVFL W.aaa
AVHL AB.alapaaaaa
AVJL AB.alapapppa
AVKL AB.plpapapap
AVL AB.prpappaap
AWAL AB.plaapapaa
AWBL AB.alpppppap
AWCL AB.plpaaaaap
BAGL AB.alppappap
BDUL AB.arppaappp
CEPDL AB.plaaaaappa
CEPVL AB.plpaappppa
FLPL AB.plapaaapad
HSNL AB.plapppappa
IL1DL AB.alapappaaa
IL1L AB.alapaappaa
IL1VL AB.alppapppaa
IL2DL AB.alapappap
IL2L AB.alapaappp
IL2VL AB.alppapppp
LUAL AB.plpppaapap
OLLL AB.alpppapaa
OLQDL AB.alapapapaa
OLQVL AB.plpaaappaa
PDEL V5L.paaa
PHAL AB.plpppaapp
```

```
PHBL AB.plapppappp
PHCL TL.pppaa
PLML AB.plapappppaa
PLNL TL.pppap
PVCL AB.plpppaapaa
PVDL V5L.paapa
PVNL TL.appp
PVPL AB.plppppaaa
PVQL AB.plapppaaa
PVWL TL.ppa
RIAL AB.alapaapaa
RIBL AB.plpaappap
RICL AB.plppaaaapp
RIFL AB.plppapaaap
RIGL AB.plppappaa
RIML AB.plppaapap
RIPL AB.alpapaaaa
RIVL AB.plpaapaaa
RMDDL AB.alpapapaa
RMDL AB.alpppapad
RMDVL AB.alppapaaa
RMEL AB.alaaaarlp
RMFL G2.al
RMGL AB.plapaaapp
RMHL G1.l
SAADL AB.alppapapa
SAAVL AB.plpaaaaaa
SABVL AB.plppapaaaa
SDQL QL.pap
SIADL AB.plpapaapa
SIAVL AB.plpapappa
SIBDL AB.plppaaaaa
SIBVL AB.plpapaapp
SMBDL AB.alpapapapp
SMBVL AB.alpapappp
SMDDL AB.plpapaaaa
SMDVL AB.alppappaa
URADL AB.plaaaaaaa
URAVL AB.plpaaapaa
URBL AB.plaapaapa
URXL AB.plaaaaappp
URYDL AB.alapapapp
URYVL AB.plpaaappp
```

9. RightLineage. This file contains just the lineage strings of the right side of the *C. elegans* nervous system.

```
ADAR AB.prapaaaapp
ADER AB.prapaaaapa
ADFR AB.praaappaa
ADLR AB.praaapaad
AFDR AB.praaaapav
AIAR AB.prppaappa
AIBR AB.praapappa
AIMR AB.prpaapppa
AINR AB.alaapaaar
AIYR AB.prpapaaap
AIZR AB.prapaaapav
ALMR AB.arpppappa
ALNR AB.prapapppppap
AQR QR.ap
ASER AB.praaapppaa
ASGR AB.praapapap
ASHR AB.prpaappaa
ASIR AB.praapapppa
ASJR AB.praaappppa
ASKR AB.praaaapppa
AUAR AB.praaapppppp
AVAR AB.alaappapa
AVBR AB.prpaapaap
AVDR AB.alaaapprl
AVER AB.praaaaaaa
AVFR P1.aaa
AVHR AB.alappapaa
AVJR AB.alapppppa
AVKR AB.prpapapap
AWAR AB.praapapaa
AWBR AB.praaappap
AWCR AB.prpaaaaap
BAGR AB.arappppap
BDUR AB.arpppappp
CEPDR AB.arpapaappa
CEPVR AB.prpaappppa
FLPR AB.prapaaapad
HSNR AB.prapppappa
IL1DR AB.alappppaaa
IL1R AB.alaappppaa
IL1VR AB.arappppaa
IL2DR AB.alappppap
IL2R AB.alaappppp
IL2VR AB.arappppppp
LUAR AB.prpppaapap
OLLR AB.praaapapaa
OLQDR AB.alapppapaa
OLQVR AB.prpaaappaa
PDER V5R.paaa
```

```
PHAR AB.prpppaapp
PHBR AB.prapppappp
PHCR TR.pppaa
PLMR AB.prapappppaa
PLNR TR.pppap
PQR QL.ap
PVCR AB.prpppaapaa
PVDR V5R.paapa
PVNR TR.appp
PVPR AB.prpppaaa
PVQR AB.prapppaaa
PVR C.aappa
PVWR TR.ppa
RIAR AB.alaapppaa
RIBR AB.prpaappap
RICR AB.prppaaaapp
RIFR AB.prppapaaap
RIGR AB.prppappaa
RIMR AB.prppaapap
RIPR AB.prappaaaa
RIR AB.prpapppaa
RIVR AB.prpaapaaa
RMDDR AB.arappapaa
RMDR AB.praaaapad
RMDVR AB.arapppaaa
RMER AB.alaaaarrp
RMFR G2.ar
RMGR AB.prapaaapp
RMHR G1.r
SAADR AB.arapppapa
SAAVR AB.prpaaaaaa
SABVR AB.prppapaaaa
SDQR QR.pap
SIADR AB.prpapaapa
SIAVR AB.prpapappa
SIBDR AB.prppaaaaa
SIBVR AB.prpapaapp
SMBDR AB.arappapapp
SMBVR AB.arappappp
SMDDR AB.prpapaaaa
SMDVR AB.arappppaa
URADR AB.arpapaaaa
URAVR AB.prpaaapaa
URBR AB.praapaapa
URXR AB.arpapaappp
URYDR AB.alapppapp
```

10. LeftEdges. This file contains a synaptic description of just the left side of the nervous system. Only part of the file is shown here, as there are 840 edges. The entire file is available on the CD.

```
ADAL AIBL 1
ADAL AVBL 1
ADAL AVDL 1
ADAL AVEL 1
ADAL RICL 1
ADAL RIML 1
ADAL RIPL 1
ADEL ADAL 1
ADEL AVAL 1
ADEL AVEL 1
ADEL AVL 1
ADEL BDUL 1
ADEL CEPDL 1
ADEL FLPL 1
ADEL IL1L 1
ADEL IL2L 1
ADEL OLLL 1
...
RICL SMDDL 1
URYDL SMDDL 1
ALNL SMDVL 1
RIAL SMDVL 1
RIML SMDVL 1
BDUL URADL 1
CEPDL URADL 1
IL2DL URADL 1
CEPVL URAVL 1
IL2VL URAVL 1
ADEL URBL 1
CEPDL URBL 1
IL2L URXL 1
RMGL URXL 1
URBL URXL 1
CEPDL URYDL 1
IL1DL URYDL 1
OLLL URYDL 1
IL1VL URYVL 1
```

11. RightEdges. This file contains a description of the right side of the nervous system of *C. elegans*. Only part of the file is shown here, as it contains 931 edges. The entire file may be viewed on the CD.

```
ADAR AIBR 1
ADAR AVBR 1
ADAR RIMR 1
ADAR RIPR 1
ADAR RIVR 1
ADER ADAR 1
ADER AVAR 1
ADER AVDR 1
ADER AVER 1
ADER AVJR 1
ADER AVKR 1
ADER CEPDR 1
ADER FLPR 1
ADER OLLR 1
ADER PVR 1
ADER RIGR 1
ADER RMDR 1
ADER SAAVR 1
...
RICR SMDDR 1
RIMR SMDDR 1
OLLR SMDVR 1
RIAR SMDVR 1
RICR SMDVR 1
RIVR SMDVR 1
BDUR URADR 1
CEPDR URADR 1
IL2DR URADR 1
CEPVR URAVR 1
IL2VR URAVR 1
CEPDR URBR 1
IL2R URBR 1
ADFR URXR 1
AUAR URXR 1
IL1R URXR 1
IL2VR URXR 1
RIR URXR 1
RMGR URXR 1
URBR URXR 1
CEPDR URYDR 1
URADR URYDR 1
```

12. LeftCorrLineage, This file is nearly identical to LeftLineage, with only neurons and lineage strings edited out that do not have corresponding neurons in the set of right neurons. The file is available on the CD.

13. RightCorrLineage. This file is nearly identical to RightLineage, with only neurons and lineage strings edited out that do not have corresponding neurons in the set of left neurons. This file is available in its entirety on the CD.

14. LeftCorrEdges. This file is nearly identical to LeftEdges, with only neurons and edges that refer to neurons which do not have a corresponding neuron in the set of right neurons edited out. This file is available in its entirety on the CD.

15. RightCorrEdges. This file is nearly identical to RightEdges, with only neurons and edges that refer to neurons which do not have a corresponding neuron in the set of left neurons edited out. This file is available in its entirety on the CD.

16. StrongComp. This file contains a list of the sets of strong components of the entire *C. elegans* synaptic network. Some strong components are single neurons.

```
Completed Reading Graph
Num Vertices is: 280
Strong Component 1:   VC6
Strong Component 2:   SDQR
Strong Component 3:   PVDR
Strong Component 4:   PLNR
Strong Component 5:   PLML
Strong Component 6:   PHCR
Strong Component 7:   IL2DR
Strong Component 8:   IL2DL
Strong Component 9:   DVB
Strong Component 10:   PHCL
Strong Component 11:   ASIR
Strong Component 12:   ASIL
Strong Component 13:   AINL
Strong Component 14:   ADAL ADAR ADEL ADER ADFL ADFR ADLL ADLR AFDL AFDR AIAL
AIAR AIBL AIBR AIML AIMR AINR AIYL AIYR AIZL AIZR ALA ALML ALMR ALNL ALNR AQR
AS1 AS11 AS2 AS3 AS4 AS5 AS6 AS9 ASEL ASER ASGL ASGR ASHL ASHR ASJL ASJR ASKL
ASKR AUAL AUAR AVAL AVAR AVBL AVBR AVDL AVDR AVEL AVER AVFL AVFR AVG AVHL
AVHR AVJL AVJR AVKL AVKR AVL AVM AWAL AWAR AWBL AWBR AWCL AWCR BAGL BAGR BDUL
BDUR CEPDL CEPDR CEPVL CEPVR DA1 DA2 DA3 DA4 DA5 DA6 DA9 DB1 DB2 DB3 DB4 DB7
DD1 DD2 DD5 DVA DVC FLPL FLPR HSNL HSNR IL1DL IL1DR IL1L IL1R IL1VL IL1VR
IL2L IL2R IL2VL IL2VR LUAL LUAR OLLL OLLR OLQDL OLQDR OLQVL OLQVR PDA PDB
PDEL PDER PHAL PHAR PHBL PHBR PLMR PLNL PQR PVCL PVCR PVDL PVM PVNL PVNR PVPL
PVPR PVQL PVQR PVR PVT PVWL PVWR RIAL RIAR RIBL RIBR RICL RICR RID RIFL RIFR
RIGL RIGR RIH RIML RIMR RIPL RIPR RIR RIS RIVL RIVR RMDDR RMDL RMDR RMDVL
RMED RMEV RMFL RMFR RMGL RMGR RMHL RMHR SAADL SAADR SAAVL SAAVR SABD SDQL
SMBDL SMBDR SMBVL SMBVR SMDDL SMDDR SMDVL SMDVR URADL URADR URAVL URAVR URBL
URBR URXL URXR URYDL URYDR URYVL URYVR VA1 VA11 VA12 VA2 VA3 VA4 VA5 VA6 VA8
VA9 VB1 VB10 VB11 VB2 VB3 VB4 VB5 VB6 VB8 VB9 VC1 VC2 VC3 VC4 VC5 VD1 VD10
VD11 VD12 VD13 VD2 VD3 VD5 VD6 VD8
Strong Component 15:   DA7
Strong Component 16:   AS8
Strong Component 17:   AS7
Strong Component 18:   VA10
Strong Component 19:   VA7
Strong Component 20:   VB7
Strong Component 21:   SIADL
Strong Component 22:   VD7
Strong Component 23:   VD9
Strong Component 24:   DD4
Strong Component 25:   DB6
Strong Component 26:   DB5
Strong Component 27:   SIADR
Strong Component 28:   SIBDL
Strong Component 29:   SIBVL
Strong Component 30:   SIAVL
Strong Component 31:   SIAVR
Strong Component 32:   RMEL
```

```
Strong Component 33:   RMER
Strong Component 34:   RMDDL RMDVR
Strong Component 35:   SIBVR
Strong Component 36:   SIBDR
Strong Component 37:   DA8
Strong Component 38:   SABVR
Strong Component 39:   SABVL
Strong Component 40:   DD6
Strong Component 41:   AS10
Strong Component 42:   DD3
Strong Component 43:   VD4
```

17. LeftStrongComp. This file contains a list of strong components identified on the left side of the nervous system.

```
Strong Component 1:   PVDL
Strong Component 2:   PLML
Strong Component 3:   PHCL
Strong Component 4:   PHAL
Strong Component 5:   IL2DL
Strong Component 6:   ASIL
Strong Component 7:   AINL
Strong Component 8:   BAGL
Strong Component 9:   ADAL ADEL ADFL ADLL AFDL AIAL AIBL AIML AIYL AIZL ALML
ALNL ASEL ASGL ASHL ASJL ASKL AUAL AVAL AVBL AVDL AVEL AVFL AVHL AVJL AVKL
AVL AWAL AWBL AWCL BDUL CEPDL CEPVL FLPL HSNL IL1DL IL1L IL1VL IL2L IL2VL
LUAL OLLL OLQVL PDEL PHBL PLNL PVCL PVNL PVPL PVQL PVWL RIAL RIBL RICL RIFL
RIGL RIML RIVL RMDL RMFL RMGL SAADL SAAVL SDQL SMBVL SMDDL SMDVL URADL URBL
URXL URYDL URYVL
Strong Component 10:   SIADL
Strong Component 11:   SIBDL
Strong Component 12:   URAVL
Strong Component 13:   RMHL
Strong Component 14:   SIAVL
Strong Component 15:   RMEL
Strong Component 16:   RMDDL
Strong Component 17:   RMDVL
Strong Component 18:   RIPL
Strong Component 19:   OLQDL
Strong Component 20:   SIBVL
Strong Component 21:   SMBDL
Strong Component 22:   SABVL
```

18. RightStrongComp. This file contains a list of the strong components identified on the right side of the nervous system.

```
Strong Component 1:   URYVR
Strong Component 2:   SDQR
Strong Component 3:   PVDR
Strong Component 4:   PLNR
Strong Component 5:   PLMR
Strong Component 6:   PHCR
Strong Component 7:   PHAR
Strong Component 8:   PHBR
Strong Component 9:   IL2DR
Strong Component 10:   ASIR
Strong Component 11:   AINR
Strong Component 12:   ADAR ADER ADFR ADLR AFDR AIAR AIBR AIMR AIYR AIZR ALMR
ALNR AQR ASER ASGR ASHR ASJR ASKR AUAR AVAR AVBR AVDR AVER AVFR AVHR AVJR
AVKR AWAR AWBR AWCR BDUR CEPDR CEPVR FLPR HSNR IL1DR IL1R IL1VR IL2R IL2VR
LUAR OLLR OLQDR OLQVR PDER PQR PVCR PVNR PVPR PVQR PVR PVWR RIAR RIBR RICR
RIFR RIGR RIMR RIPR RIR RIVR RMDR RMFR RMGR SAADR SAAVR SMBVR SMDDR SMDVR
URADR URAVR URBR URXR URYDR
Strong Component 13:   SIADR
Strong Component 14:   RMDVR
Strong Component 15:   SIBVR
Strong Component 16:   SIAVR
Strong Component 17:   SIBDR
Strong Component 18:   RMHR
Strong Component 19:   RMER
Strong Component 20:   RMDDR
Strong Component 21:   SMBDR
Strong Component 22:   BAGR
Strong Component 23:   SABVR
```

19. Sensory.txt

```
40
ADEL
ADER
ADFL
ADFR
ADLL
ADLR
AFDL
AFDR
ALML
ALMR
ASEL
ASER
ASGL
ASGR
ASHL
ASHR
ASIL
ASIR
ASJL
ASJR
ASKL
ASKR
AVM
AWAL
AWAR
AWBL
AWBR
AWCL
AWCR
PDEL
PDER
PHAL
PHAR
PHBL
PHBR
PHCL
PHCR
PLML
PLMR
PVM
```

20. Motor.txt

```
AS1
AS2
AS3
AS4
AS5
AS6
AS7
AS8
AS9
AS10
AS11
DA1
DA2
DA3
DA4
DA5
DA6
DA7
DA8
DA9
DB1
DB3
DB2
DB4
DB5
DB6
DB7
DD1
DD2
DD3
DD4
DD5
DD6
PDA
PDB
RMDDL
RMDDR
RMDL
RMDR
RMDVL
RMDVR
RMED
RMEL
RMER
RMEV
RMFL
RMFR
RMHL
RMHR
URADL
URADR
```

URAVL
URAVR
VA1
VA2
VA3
VA4
VA5
VA6
VA7
VA8
VA9
VA10
VA11
VA12
VB1
VB2
VB3
VB4
VB5
VB6
VB7
VB8
VB9
VB10
VB11
VC1
VC2
VC3
VC4
VC5
VC6
VD1
VD2
VD3
VD4
VD5
VD6
VD7
VD8
VD9
VD10
VD11
VD12
VD13

21. NeuronPaths – this file contains paths between a set of input neurons (e.g. sensory neurons) and a set of output neurons (e.g. motor neurons). The entire file is too large to be presented. Only portions of this file are presented, the file is available in its entirety on disk.

```
Begining neuron is: ADEL
Ending neuron is: AS1
Path length is: 2
Path is: ADEL  AVAR  AS1
. . .
Begining neuron is: ADEL
Ending neuron is: VA7
Path length is: 3
Path is: ADEL  ADAL  AVBL  VA7
. . .
Begining neuron is: ADLR
Ending neuron is: VC4
Path length is: 5
Path is: ADLR  ASER  AIAL  HSNL  VC5  VC4

Begining neuron is: ADLR
Ending neuron is: VC5
Path length is: 4
Path is: ADLR  ASER  AIAL  HSNL  VC5
. . .
Begining neuron is: ASEL
Ending neuron is: VB9
Path length is: 5
Path is: ASEL  ADFR  PVPR  PVCL  VB8  VB9
. . .
Begining neuron is: ASGR
Ending neuron is: VB2
Path length is: 4
Path is: ASGR  AIAR  ADAR  AVBL  VB2
. . .
Begining neuron is: ASIR
Ending neuron is: VB9
Path length is: 5
Path is: ASIR  AIAR  ADLR  PVCL  VB8  VB9
. . .
Begining neuron is: ASJR
Ending neuron is: URADR
Path length is: 3
Path is: ASJR  HSNR  BDUR  URADR

Begining neuron is: ASJR
Ending neuron is: URAVL
Path length is: 5
Path is: ASJR  HSNR  AIBR  RIAL  CEPVL  URAVL
. . .
Begining neuron is: ASKR
Ending neuron is: VD13
Path length is: 4
```

Path is: ASKR AIAR ADLR AVAR VD13

22. Neurongenessynapses

This file is used by develstages, pathgenes, pathsgenes, neuronenvgenes, and pathgenes pairs. It contains information on neurons, lineage strings, neuron categories, genes of neurons, and synapses.

```
280
ADAL  AB.plapaaaapp     IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 eat-4 ina-1 lin-35      lin-53      sax-7 set-2 unc-86      end



ADAR  AB.prapaaaapp     IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 eat-4 ina-1 lin-35      lin-53      sax-7 set-2 unc-86      end



ADEL  AB.plapaaaapa     SN    AD    D     cam-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      chd-3 che-2 che-3 dat-1 dop-2 egl-36      gpa-14      ida-1
      ina-1 lin-35      lin-53      osm-6 set-2 unc-103     end



ADER  AB.prapaaaapa     SN    AD    D     cam-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      chd-3 che-2 che-3 dat-1 dop-2 egl-36      gpa-14      ida-1
      ina-1 lin-35      lin-53      osm-6 set-2 unc-103     end



ADFL  AB.alpppppaa      SN    CH    A     cat-1 cav-1 cdc-42      ced-10
      chd-3 che-3 csk-1 fbxb-103    gpa-10      gpa-13      gpa-3 ina-1 kvs-1
      kvs-1 lin-11      lin-11      lin-35      lin-53      lst-1 mps-1 mps-1
      ncs-1 ncs-1 nlp-3 ocr-2 odr-3 odr-3 osm-3 osm-6 osm-9 ptp-3 set-2 srb-6
      src-1 srd-1 tax-6 tax-6 tax-6 tph-1 end

ADFR  AB.praaappaa      SN    CH    A     cat-1 cav-1 cdc-42      ced-10
      chd-3 che-3 csk-1 fbxb-103    gpa-10      gpa-13      gpa-3 ina-1 kvs-1
      kvs-1 lin-11      lin-11      lin-35      lin-53      lst-1 mps-1 mps-1
      ncs-1 ncs-1 nlp-3 ocr-2 odr-3 odr-3 osm-3 osm-6 osm-9 ptp-3 set-2 srb-6
      src-1 srd-1 tax-6 tax-6 tax-6 tph-1 end

ADLL  AB.alpppaad       SN    CH    A     cam-1 cav-1 cdc-42      ced-10
      ceh-23      ceh-32      chd-3 che-3 cog-1 fbxb-103    flp-21      gpa-1
      gpa-11      gpa-15      gpa-3 gpc-1 hlh-2 ina-1 kvs-1 kvs-1 lin-11
      lin-11      lin-35      lin-53      nhr-79      nlp-10      nlp-7 nlp-8
      ocr-1 ocr-2 osm-3 osm-6 osm-9 osm-9 osm-9 osm-9 ptp-3 qui-1 set-2 sgk-1
      srb-6 sre-1 srh-220     sro-1 tax-6 tax-6 tax-6 ttx-3 unc-103     ver-2
      end
ADLR  AB.praaapaad      SN    CH    A     cam-1 cav-1 cdc-42      ced-10
      ceh-23      ceh-32      chd-3 che-3 cog-1 fbxb-103    flp-21      gpa-1
      gpa-11      gpa-15      gpa-3 gpc-1 hlh-2 ina-1 kvs-1 kvs-1 lin-11
      lin-11      lin-35      lin-53      nhr-79      nlp-10      nlp-7 nlp-8
```

```
      ocr-1 ocr-2 osm-3 osm-6 osm-9 osm-9 osm-9 osm-9 ptp-3 qui-1 set-2 sgk-1
      srb-6 sre-1 srh-220     sro-1 tax-6 tax-6 tax-6 ttx-3 unc-103     ver-2
      end
AFDL  AB.alpppapav      SN    CH    A     cat-1 cav-1 cdc-42      ced-10
      chd-3 che-3 csk-1 fbxb-103    gpa-10      gpa-13      gpa-3 ina-1 kvs-1
      kvs-1 lin-11      lin-11      lin-35      lin-53      lst-1 mps-1 mps-1
      ncs-1 ncs-1 nlp-3 ocr-2 odr-3 odr-3 osm-3 osm-6 osm-9 ptp-3 set-2 srb-6
      src-1 srd-1 tax-6 tax-6 tax-6 tph-1 end

AFDR  AB.praaaapav      SN    CH    A     cat-1 cav-1 cdc-42      ced-10
      chd-3 che-3 csk-1 fbxb-103    gpa-10      gpa-13      gpa-3 ina-1 kvs-1
      kvs-1 lin-11      lin-11      lin-35      lin-53      lst-1 mps-1 mps-1
      ncs-1 ncs-1 nlp-3 ocr-2 odr-3 odr-3 osm-3 osm-6 osm-9 ptp-3 set-2 srb-6
      src-1 srd-1 tax-6 tax-6 tax-6 tph-1 end

AIAL  AB.plppaappa      IN    A     x     cav-1 cdc-42      ced-10      chd-3
      cog-1 fbxb-103    glr-2 gpa-2 gpa-2 ina-1 lin-35      lin-53      mgl-1
      ptp-3 set-2 ttx-3 end


AIAR  AB.prppaappa      IN    A     x     cav-1 cdc-42      ced-10      chd-3
      cog-1 fbxb-103    glr-2 gpa-2 gpa-2 ina-1 lin-35      lin-53      mgl-1
      ptp-3 set-2 ttx-3 end


AIBL  AB.plaapappa      IN    A     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ggr-1 glr-1 glr-1 glr-2 glr-5 ina-1 lin-35      lin-53
      mgl-2 odr-2 ptp-3 set-2 tax-6 unc-8 end


AIBR  AB.praapappa      IN    A     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ggr-1 glr-1 glr-1 glr-2 glr-5 ina-1 lin-35      lin-53
      mgl-2 odr-2 ptp-3 set-2 tax-6 unc-8 end


AIML  AB.plpaapppa      IN    R     x     cam-1 cat-1 cav-1 cdc-42      ced-
10    chd-3 fbxb-103    hid-1 ina-1 lin-35      lin-53      ptp-3 set-2 tax-6
      tax-6 tph-1 unc-86      zig-3 end


AIMR  AB.prpaapppa      IN    R     x     cam-1 cat-1 cav-1 cdc-42      ced-
10    chd-3 fbxb-103    hid-1 ina-1 let-60      lin-35      lin-53      ptp-3
      set-2 tax-6 tax-6 tph-1 unc-86      zig-3 end


AINL  AB.alaaaalal      IN    R     x     cam-1 cav-1 cdc-42      ced-10
      ceh-10      chd-3 eat-4 fbxb-103    ina-1 lin-35      lin-53      ptp-3
      set-2 unc-103     unc-42      unc-8 end



AINR  AB.alaapaaar      IN    R     x     cam-1 cav-1 cdc-42      ced-10
      ceh-10      chd-3 eat-4 fbxb-103    ina-1 lin-35      lin-53      ptp-3
```

```
      set-2 unc-103     unc-42      unc-8 end


AIYL  AB.plpapaaap      IN    A     x     cam-1 cat-1 cav-1 cdc-42      ced-
10    ceh-10      ceh-23      chd-3 cnb-1 fbxb-103    flp-18      hen-1 ina-1
      kal-1 kin-29      lin-35      lin-53      mgl-1 ncs-1 ncs-1 ptp-3 ser-2
      set-2 sra-11      tax-6 ttx-3 ttx-3 ttx-3 wrk-1 end


AIYR  AB.prpapaaap      IN    A     x     cam-1 cat-1 cav-1 cdc-42      ced-
10    ceh-10      ceh-23      chd-3 cnb-1 fbxb-103    flp-18      hen-1 ina-1
      kal-1 kin-29      let-60      lin-35      lin-53      mgl-1 ncs-1 ncs-1
      ptp-3 ser-2 set-2 sra-11      tax-6 ttx-3 ttx-3 ttx-3 wrk-1 end


AIZL  AB.plapaaapav     IN    A     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 cnb-1 gpa-3 ina-1 kal-1 kin-29      lin-11      lin-11      lin-
35    lin-53      mec-3 odr-2 ser-2 set-2 tax-6 unc-8 unc-86      unc-86
      end


AIZR  AB.prapaaapav     IN    A     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 cnb-1 gpa-3 ina-1 kal-1 kin-29      lin-11      lin-11      lin-
35    lin-53      mec-3 odr-2 ser-2 set-2 tax-6 unc-8 unc-86      unc-86
      end


ALA   AB.alapppaaa      IN    DC    x     cav-1 cdc-42      ced-10      ceh-
10    ceh-14      ceh-17      chd-3 fbxb-103    gly-18      gpa-14      ida-1
      ida-1 ida-1 ina-1 lin-35      lin-53      ptp-3 sax-3 set-2 sra-10
      tba-2 unc-103     unc-53      unc-53      ver-3 end


ALML  AB.arppaappa      SN    T     x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-1 daf-5 deg-3 dop-1 dop-1 dyn-1 eat-4 eat-4 egl-2 egl-21
      egl-3 egl-46      glr-8 goa-1 ina-1 jkk-1 jnk-1 lin-14      lin-14
      lin-35      lin-53      mec-10      mec-2 mec-3 mec-3 mec-4 mec-6 mec-7
      mec-7 mec-7 mec-7 mec-8 mig-2 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1 ptl-1
      sax-7 set-2 tag-97      tba-1 tba-1 tba-1 tol-1 unc-32      unc-73
      unc-86      unc-97      end
ALMR  AB.arpppappa      SN    T     x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-1 daf-5 deg-3 dop-1 dop-1 dyn-1 eat-4 eat-4 egl-2 egl-21
      egl-3 egl-46      glr-8 goa-1 ina-1 jkk-1 jnk-1 lin-14      lin-14
      lin-35      lin-53      mec-10      mec-2 mec-3 mec-3 mec-4 mec-6 mec-7
      mec-7 mec-7 mec-7 mec-8 mig-2 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1 ptl-1
      sax-7 set-2 tag-97      tba-1 tba-1 tba-1 tol-1 unc-32      unc-73
      unc-86      unc-97      end
ALNL  AB.plapappppap    IN    AL    x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    cha-1 chd-3 dop-1 eat-20      eat-20      egl-2 egl-36      egl-36
      gcy-35      gcy-35      gpa-10      ina-1 lad-2 lin-35      lin-53
      mec-7 odr-2 ser-2 set-2 tol-1 unc-103     unc-17      unc-32      unc-
```

53 unc-86 end

ALNR AB.prapapppppap IN AL x ahr-1 cam-1 cav-1 cdc-42 ced-10 cha-1 chd-3 dop-1 eat-20 eat-20 egl-2 egl-36 egl-36 gcy-35 gcy-35 gpa-10 ina-1 lad-2 lin-35 lin-53 mec-7 odr-2 ser-2 set-2 tol-1 unc-103 unc-17 unc-32 unc-53 unc-86 end

AQR ABprapapaaQRap IN R x ahr-1 che-2 che-3 cki-1 egl-2 egl-2 gcy-32 gcy-34 gcy-35 gcy-35 gcy-36 gcy-36 gcy-36 gcy-37 gpa-8 npr-1 osm-6 tax-2 tax-2 end

AS1 ABplapaappP1apa MN VC x hmr-1 unc-53 unc-53 end

AS2 ABplapaappP2apa MN VC x exc-7 hmr-1 unc-53 unc-53 end

AS3 ABplappaaaP3apa MN VC x hmr-1 unc-53 unc-53 end

AS4 ABplappaaaP4apa MN VC x exc-7 hmr-1 unc-53 unc-53 end

AS5 ABplappaapP5apa MN VC x hmr-1 unc-53 unc-53 end

AS6 ABplappaapP6apa MN VC x hmr-1 unc-53 unc-53 end

AS7 ABplappappP7apa MN VC x hmr-1 unc-53 unc-53 end

AS8 ABplappappP8apa MN VC x hmr-1 unc-53 unc-53 end

AS9 ABplapapapP9apa MN VC x hmr-1 unc-53 unc-53 end

AS10 ABplapapap10apa MN VC x hmr-1 unc-53 unc-53 end

AS11 ABplapappa11apa MN VC x hmr-1 lin-29 unc-42 unc-53 unc-53 end

ASEL AB.alpppppaa SN CH A cav-1 cdc-42 ced-10 ceh-23 ceh-36 ceh-36 chd-3 che-1 che-3 cog-1 csk-1 egl-2 egl-2 egl-4 fbxb-103 flp-21 gcy-6 gcy-7 gpa-3 hen-1 ina-1 kvs-1 kvs-1 lim-6 lin-35 lin-53 lsy-6 mps-1 mps-1 ncs-1 ncs-1 nlp-14 nlp-3 nlp-7 npr-1 osm-3 osm-6 osm-9 pef-1 set-2 src-1 tax-2 tax-2 tax-4 tax-6 tax-6 unc-37 unc-5 unc-5 end

ASER AB.praaapppaa SN CH A cav-1 cdc-42 ced-10 ceh-23 ceh-36 ceh-36 chd-3 che-1 che-3 cog-1 csk-1 egl-2 egl-2 egl-4 fbxb-103 flp-21 gcy-5 gpa-3 hen-1 ina-1 kvs-1 kvs-1 lin-35 lin-53 mps-1 mps-1 ncs-1 ncs-1 nlp-14 nlp-3 nlp-7 npr-1 osm-3 osm-6 osm-9 pef-1 set-2 src-1 tax-2 tax-2 tax-4 tax-6 tax-6 unc-37 unc-5 unc-5 end

ASGL AB.plaapapap SN CH A bra-1 cav-1 cdc-42 ced-10 ceh-23 chd-3 che-3 fbxb-103 gpa-3 ida-1 ina-1 kin-29 kvs-1 kvs-1 lim-6 lin-11 lin-35 lin-53 lst-1 mps-1 mps-1 ncs-1 ncs-1 nhr-83 npr-1 odr-2 osm-3 osm-6 osm-9 ptp-3 set-2 tax-2 tax-2 tax-4 unc-30 end

```
ASGR  AB.praapapap      SN    CH    A     bra-1 cav-1 cdc-42      ced-10
      ceh-23      chd-3 che-3 fbxb-103    gpa-3 ida-1 ina-1 kin-29      kvs-1
      kvs-1 lim-6 lin-11      lin-35      lin-53      lst-1 mps-1 mps-1 ncs-1
      ncs-1 nhr-83      npr-1 odr-2 osm-3 osm-6 osm-9 ptp-3 set-2 tax-2 tax-2
      tax-4 unc-30      end


ASHL  AB.plpaappaa      SN    CH    A     cam-1 cam-1 cav-1 cdc-42      ced-
10    ceh-23      chd-3 che-1 che-3 eat-4 egl-3 egl-4 fbxb-103    flp-21
      gpa-1 gpa-11      gpa-13      gpa-14      gpa-15      gpa-3 gpc-1 grk-2
      hlh-2 ina-1 kin-29      kvs-1 kvs-1 lin-11      lin-35      lin-53
      mps-1 mps-1 nhr-79      nlp-15      nlp-3 npr-1 ocr-2 odr-3 odr-3 osm-
10    osm-10      osm-10      osm-3 osm-6 osm-9 ptp-3 qui-1 set-2 sra-6 srb-6
      tax-6 tax-6 tax-6 unc-42      unc-8 unc-8 end

ASHR  AB.prpaappaa      SN    CH    A     cam-1 cam-1 cav-1 cdc-42      ced-
10    ceh-23      chd-3 che-1 che-3 eat-4 egl-3 egl-4 fbxb-103    flp-21
      gpa-1 gpa-11      gpa-13      gpa-14      gpa-15      gpa-3 gpc-1 grk-2
      hlh-2 ina-1 kin-29      kvs-1 kvs-1 lin-11      lin-35      lin-53
      mps-1 mps-1 nhr-79      nlp-15      nlp-3 npr-1 ocr-2 odr-3 odr-3 osm-
10    osm-10      osm-10      osm-3 osm-6 osm-9 ptp-3 qui-1 set-2 sra-6 srb-6
      tax-6 tax-6 tax-6 unc-42      unc-8 unc-8 end

ASIL  AB.plaapapppa     SN    CH    A     bra-1 cam-1 cav-1 cdc-42      ced-
10    ceh-23      chd-3 che-3 daf-11      daf-28      daf-7 daf-7 gcs-1 gpa-1
      gpa-10      gpa-14      gpa-3 gpa-4 gpa-5 gpa-6 gpc-1 ida-1 ina-1 kal-1
      kin-29      lin-35      lin-53      nlp-1 nlp-14      nlp-18      nlp-
24    nlp-27      nlp-5 nlp-6 nlp-7 nlp-9 odr-1 osm-10      osm-10      osm-
10    osm-3 osm-6 osm-9 set-2 sgk-1 skn-1 sra-6 srd-1 srd-1 str-2 str-2 str-3
      tax-2 tax-2 tax-2 tax-4 tax-6 tax-6 tax-6 ttx-3 unc-3 unc-3 wrk-1 zig-3
      zig-4 end
ASIR  AB.praapapppa     SN    CH    A     bra-1 cam-1 cav-1 cdc-42      ced-
10    ceh-23      chd-3 che-3 daf-11      daf-28      daf-7 daf-7 gcs-1 gpa-1
      gpa-10      gpa-14      gpa-3 gpa-4 gpa-5 gpa-6 gpc-1 ida-1 ina-1 kal-1
      kin-29      lin-35      lin-53      nlp-1 nlp-14      nlp-18      nlp-
24    nlp-27      nlp-5 nlp-6 nlp-7 nlp-9 odr-1 osm-10      osm-10      osm-
10    osm-3 osm-6 osm-9 set-2 sgk-1 skn-1 sra-6 srd-1 srd-1 str-2 str-2 str-3
      tax-2 tax-2 tax-2 tax-4 tax-6 tax-6 tax-6 ttx-3 unc-3 unc-3 wrk-1 zig-3
      zig-4 end
ASJL  AB.alpppppppa     SN    CH    A     cav-1 cdc-42      ced-10      chd-3
      che-3 cog-1 daf-11      daf-28      fbxb-103    gpa-1 gpa-10      gpa-
14    gpa-3 gpa-9 gpc-1 ina-1 kin-29      lin-35      lin-53      nlp-3 odr-1
      osm-3 osm-6 osm-9 set-2 sre-1 tax-2 tax-2 tax-4 tbx-2 unc-103     end


ASJR  AB.praaappppa     SN    CH    A     cav-1 cdc-42      ced-10      chd-3
      che-3 cog-1 daf-11      daf-28      fbxb-103    gpa-1 gpa-10      gpa-
14    gpa-3 gpa-9 gpc-1 ina-1 kin-29      lin-35      lin-53      nlp-3 odr-1
      osm-3 osm-6 osm-9 set-2 sre-1 tax-2 tax-2 tax-4 tbx-2 unc-103     end

ASKL  AB.alpppapppa     SN    CH    A     ahr-1 bra-1 cam-1 cav-1 cdc-42
      ced-10      chd-3 che-3 daf-11      eat-4 egl-4 fbxb-103    gpa-14
```

```
      gpa-15      gpa-3 ida-1 ina-1 kin-29      kvs-1 kvs-1 lin-35      lin-
53    nlp-10      nlp-14      nlp-8 odr-1 osm-3 osm-6 osm-9 set-2 sgk-1 sra-7
      sra-9 srg-2 srg-8 tax-2 tax-2 tax-2 tax-4 tax-6 tax-6 tax-6 unc-103
      zig-4 zig-5 end
ASKR  AB.praaaapppa     SN    CH    A     ahr-1 bra-1 cam-1 cav-1 cdc-42
      ced-10      chd-3 che-3 daf-11      eat-4 egl-4 fbxb-103    gpa-14
      gpa-15      gpa-3 ida-1 ina-1 kin-29      kvs-1 kvs-1 lin-35      lin-
53    nlp-10      nlp-14      nlp-8 odr-1 osm-3 osm-6 osm-9 set-2 sgk-1 sra-7
      sra-9 srg-2 srg-8 tax-2 tax-2 tax-2 tax-4 tax-6 tax-6 tax-6 unc-103
      zig-4 zig-5 end
AUAL  AB.alpppppppp     IN    A     x     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 che-1 csk-1 dop-1 eat-4 fbxb-103    flr-4 glr-4 ida-1
      ina-1 lin-35      lin-53      npr-1 ser-2 set-2 src-1 tax-6 tax-6 tax-6
      end

AUAR  AB.praaapppppp     IN    A     x     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 che-1 csk-1 dop-1 eat-4 fbxb-103    flr-4 glr-4 ida-1
      ina-1 lin-35      lin-53      npr-1 ser-2 set-2 src-1 tax-6 tax-6 tax-6
      end

AVAL  AB.alppaaapa      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      chd-3 csk-1 fbxb-103    flp-1 flp-18      ggr-2 glr-1 glr-1 glr-2 glr-4
      glr-5 gpa-14      ina-1 lin-11      lin-35      lin-53      nmr-1 nmr-1
      nmr-1 nmr-2 pef-1 ptp-3 rig-3 sax-3 set-2 src-1 tax-6 tax-6 unc-103
      unc-42      unc-6 unc-8 unc-8 end

AVAR  AB.alaappapa      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      chd-3 csk-1 fbxb-103    flp-1 flp-18      ggr-2 glr-1 glr-1 glr-2 glr-4
      glr-5 gpa-14      ina-1 lin-11      lin-35      lin-53      nmr-1 nmr-1
      nmr-1 nmr-2 pef-1 ptp-3 rig-3 sax-3 set-2 src-1 tax-6 tax-6 unc-103
      unc-42      unc-6 unc-8 unc-8 end

AVBL  AB.plpaapaap      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      chd-3 egl-3 fbxb-103    ggr-2 glr-1 glr-1 glr-5 ina-1 lin-35      lin-
53    pef-1 ptp-3 sax-3 set-2 sra-11      unc-40      unc-6 unc-8 unc-8 end


AVBR  AB.prpaapaap      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      chd-3 egl-3 fbxb-103    ggr-2 glr-1 glr-1 glr-5 ina-1 lin-35      lin-
53    pef-1 ptp-3 sax-3 set-2 sra-11      unc-40      unc-6 unc-8 unc-8 end


AVDL  AB.alaaapalr      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      cfi-1 chd-3 deg-1 egl-3 fbxb-103    glr-1 glr-1 glr-2 glr-5 ina-1 lin-
35    lin-53      nmr-1 nmr-1 nmr-2 ptp-3 sax-3 set-2 tol-1 unc-103     unc-
42    unc-8 unc-8 end

AVDR  AB.alaaapprl      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      cfi-1 chd-3 deg-1 egl-3 fbxb-103    glr-1 glr-1 glr-2 glr-5 ina-1 lin-
35    lin-53      nmr-1 nmr-1 nmr-2 ptp-3 sax-3 set-2 tol-1 unc-103     unc-
42    unc-8 unc-8 end
```

```
AVEL  AB.alpppaaaa      IN    VC    x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 egl-36      fbxb-103    flp-1 glr-1 glr-1 glr-2 glr-5 ina-1 lin-
11    lin-35      lin-53      mgl-2 nmr-1 nmr-1 nmr-1 nmr-2 opt-3 ptp-3 set-2
      tax-6 tax-6 tax-6 unc-42      unc-8 end

AVER  AB.praaaaaaa      IN    VC    x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 egl-36      fbxb-103    flp-1 glr-1 glr-1 glr-2 glr-5 ina-1 lin-
11    lin-35      lin-53      mgl-2 nmr-1 nmr-1 nmr-1 nmr-2 opt-3 ptp-3 set-2
      tax-6 tax-6 tax-6 unc-42      unc-8 end

AVFL  AB.prappaapWaaa   IN    VC    R     pag-3 pag-3 unc-4 end



AVFR  ABplapaappP1aaa   IN    VC    R     pag-3 pag-3 unc-4 end



AVG   AB.prpapppap      IN    VC    x     cav-1 cdc-42      ced-10      chd-3
      deg-1 deg-3 fbxb-103    glr-1 glr-1 glr-1 glr-2 ina-1 lin-11      lin-
11    lin-35      lin-53      nmr-1 nmr-1 nmr-2 odr-2 ptp-3 set-2 unc-6 unc-
71    unc-8 end

AVHL  AB.alapaaaaa      IN    VC    R     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 fbxb-103    ggr-1 glr-4 ida-1 ina-1 lin-11      lin-
11    lin-35      lin-53      ptp-3 ser-2 set-2 unc-103     unc-42      end



AVHR  AB.alappapaa      IN    VC    R     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 fbxb-103    ggr-1 glr-4 ida-1 ina-1 lin-11      lin-
11    lin-35      lin-53      ptp-3 ser-2 set-2 unc-103     unc-42      end



AVJL  AB.alapapppa      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      ceh-10      chd-3 daf-14      eat-4 egl-4 fbxb-103    glr-1 ida-1 ina-1
      lin-11      lin-11      lin-35      lin-53      ptp-3 set-2 unc-103
      unc-42      unc-8 end


AVJR  AB.alapppppa      IN    VC    x     cam-1 cav-1 cdc-42      ced-10
      ceh-10      chd-3 daf-14      eat-4 egl-4 fbxb-103    glr-1 ida-1 ina-1
      lin-11      lin-11      lin-35      lin-53      ptp-3 set-2 unc-103
      unc-42      unc-8 end


AVKL  AB.plpapapap      IN    VC    R     cam-1 cav-1 cdc-42      ced-10
      chd-3 dbl-1 fax-1 fbxb-103    flp-1 glr-5 ina-1 let-60      lin-35
      lin-53      ncs-1 ncs-1 ptp-3 set-2 tax-6 tax-6 unc-42      end
```

```
AVKR  AB.prpapapap      IN    VC    R     cam-1 cav-1 cdc-42      ced-10
      chd-3 dbl-1 fax-1 fbxb-103    flp-1 glr-5 ina-1 let-60      lin-35
      lin-53      ncs-1 ncs-1 ptp-3 set-2 tax-6 tax-6 unc-42      vab-8 end


AVL   AB.prpappaap      IN    VC    R     cav-1 cdc-42      ced-10      chd-3
      egl-36      egl-36      fbxb-103    ina-1 lim-6 lin-35      lin-53
      ptp-3 set-2 smp-1 unc-25      unc-25      unc-25      unc-47      unc-
47    unc-47      end


AVM   ABprapapaaQRpaa   SN    T     x     ahr-1 cam-1 dop-1 eat-4 eat-4 egl-
21    egl-3 egl-46      gcy-35      gcy-35      goa-1 gpa-16      jkk-1 jnk-1
      mab-21      mec-10      mec-2 mec-3 mec-3 mec-4 mec-6 mec-7 mec-7 mec-7
      mec-7 mec-8 mtd-1 pag-3 pat-4 ptl-1 sax-7 scc-1 tba-1 tba-1 tba-1 tol-1
      unc-2 unc-32      unc-40      unc-97      end

AWAL  AB.plaapapaa      SN    A     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    gpa-5 gpa-5 gpa-6 ina-1 kin-29      lim-6 lin-11      lin-
35    lin-53      ncs-1 ncs-1 ocr-1 ocr-2 odr-10      odr-3 odr-3 odr-3 odr-7
      osm-6 osm-9 osm-9 osm-9 ptp-3 set-2 sra-13      tax-6 tax-6 end


AWAR  AB.praapapaa      SN    A     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    gpa-5 gpa-5 gpa-6 ina-1 kin-29      lim-6 lin-11      lin-
35    lin-53      ncs-1 ncs-1 ocr-1 ocr-2 odr-10      odr-3 odr-3 odr-3 odr-7
      osm-6 osm-9 osm-9 osm-9 ptp-3 set-2 sra-13      tax-6 tax-6 end


AWBL  AB.alpppppap      SN    A     x     ace-2 cav-1 cdc-42      ced-10
      ceh-37      chd-3 daf-11      fbxb-103    gpc-1 ina-1 kin-29      lim-4
      lim-4 lin-35      lin-53      lst-1 ncs-1 ncs-1 nlp-3 nlp-9 odr-1 odr-1
      odr-3 odr-3 odr-3 osm-6 pef-1 ptp-3 set-2 str-1 tax-2 tax-2 tax-4 tbx-2
      end

AWBR  AB.praaappap      SN    A     x     ace-2 cav-1 cdc-42      ced-10
      ceh-37      chd-3 daf-11      fbxb-103    gpc-1 ina-1 kin-29      lim-4
      lim-4 lin-35      lin-53      lst-1 ncs-1 ncs-1 nlp-3 nlp-9 odr-1 odr-1
      odr-3 odr-3 odr-3 osm-6 pef-1 ptp-3 set-2 str-1 tax-2 tax-2 tax-4 tbx-2
      end

AWCL  AB.plpaaaaap      SN    A     x     ace-2 bra-1 cav-1 cdc-42      ced-
10    ceh-23      ceh-36      ceh-36      chd-3 daf-11      egl-2 egl-2 egl-4
      fbxb-103    gpa-13      gpa-2 ina-1 kin-29      kvs-1 kvs-1 lin-35
      lin-53      mps-1 mps-1 ncs-1 ncs-1 nlp-1 nsy-1 odr-1 odr-1 odr-1 odr-3
      odr-3 odr-3 osm-6 osm-9 pef-1 ptp-3 set-2 sra-13      str-2 str-2 tax-2
      tax-2 tax-2 tax-4 tax-6 tax-6 tbx-2 unc-2 end
AWCR  AB.prpaaaaap      SN    A     x     ace-2 bra-1 cav-1 cdc-42      ced-
10    ceh-23      ceh-36      ceh-36      chd-3 daf-11      egl-2 egl-2 egl-4
      fbxb-103    gpa-13      gpa-2 ina-1 kin-29      kvs-1 kvs-1 lin-35
      lin-53      mps-1 mps-1 ncs-1 ncs-1 nlp-1 nsy-1 odr-1 odr-1 odr-1 odr-3
```

```
      odr-3 odr-3 osm-6 osm-9 pef-1 ptp-3 set-2 sra-13      str-2 str-2 tax-2
      tax-2 tax-2 tax-4 tax-6 tax-6 tbx-2 unc-2 end
BAGL  AB.alppappap      IN    H     x     cav-1 cdc-42      ced-10      ceh-
23    chd-3 csk-1 eat-20      egl-2 egl-2 fbxb-103    gcy-33      ina-1 lin-
35    lin-53      ncs-1 ncs-1 nhr-83      nlp-3 pef-1 ptp-3 set-2 src-1 tax-2
      tax-2 tax-4 trp-1 zig-4 end


BAGR  AB.arappppap      IN    H     x     cav-1 cdc-42      ced-10      ceh-
23    chd-3 csk-1 eat-20      egl-2 egl-2 fbxb-103    gcy-33      ina-1 lin-
35    lin-53      ncs-1 ncs-1 nhr-83      nlp-3 pef-1 ptp-3 set-2 src-1 tax-2
      tax-2 tax-4 trp-1 zig-4 end


BDUL  AB.arppaappp      IN    EC    R     ahr-1 cam-1 cav-1 cdc-42      ced-
10    ceh-14      ceh-14      chd-3 egl-21      gcy-35      glr-8 goa-1 gpa-
16    ina-1 jkk-1 jnk-1 lin-14      lin-14      lin-35      lin-53      mec-7
      mec-7 nlp-1 nlp-15      pag-3 pag-3 pkc-1 ser-2 set-2 slt-1 unc-53
      unc-8 unc-86      end
BDUR  AB.arpppappp      IN    EC    R     ahr-1 cam-1 cav-1 cdc-42      ced-
10    ceh-14      ceh-14      chd-3 egl-21      gcy-35      glr-8 goa-1 gpa-
16    ina-1 jkk-1 jnk-1 lin-14      lin-14      lin-35      lin-53      mec-7
      mec-7 nlp-1 nlp-15      pag-3 pag-3 pkc-1 ser-2 set-2 slt-1 unc-53
      unc-8 unc-86      end
CEPDL AB.plaaaaappa     IN    H     D     ace-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      ceh-10      chd-3 che-2 dat-1 dop-2 fax-1 fbxb-103    ina-1
      lin-35      lin-53      osm-6 set-2 sre-37      end

CEPDR AB.arpapaappa     IN    H     D     ace-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      ceh-10      chd-3 che-2 dat-1 dop-2 fax-1 fbxb-103    ina-1
      lin-35      lin-53      mdl-1 osm-6 set-2 sre-37      end

CEPVL AB.plpaappppa     IN    H     D     ace-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      chd-3 che-2 dat-1 dop-2 ina-1 lin-35      lin-53      osm-6
      set-2 end

CEPVR AB.prpaappppa     IN    H     D     ace-1 cat-2 cat-2 cav-1 cdc-42
      ced-10      chd-3 che-2 dat-1 dop-2 ina-1 lin-35      lin-53      mdl-1
      osm-6 set-2 end

DA1   AB.prppapaap      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129     unc-4
      unc-5 unc-53      unc-53      end

DA2   AB.plppapapa      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129     unc-
53    unc-53      end

DA3   AB.prppapapa      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-53      unc-53      end
```

```
DA4   AB.plppapapp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-53      unc-53      end

DA5   AB.prppapapp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-53      unc-53      end

DA6   AB.plpppaaap      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-53      unc-53      end

DA7   AB.prpppaaap      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-53      unc-53      end

DA8   AB.prpapappp      MN    VC    DR    cav-1 cdc-42      ced-10      ceh-
17    chd-3 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-53
      unc-53      end

DA9   AB.plpppaaaa      MN    VC    DR    avr-15      cav-1 cdc-42      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      mig-13      ptp-3 ser-2
      set-2 unc-53      unc-53      end

DB1   AB.plpaaaapp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      unc-5 vab-7 end

DB2   AB.arappappa      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      vab-7 end

DB3   AB.prpaaaapp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      vab-7 end

DB4   AB.prpappapp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      vab-7 end

DB5   AB.plpapappp      MN    VC    DR    cav-1 cdc-42      ced-10      ceh-
17    chd-3 dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-
129   vab-7 end

DB6   AB.plppaappp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      vab-7 end

DB7   AB.prppaappp      MN    VC    DR    cav-1 cdc-42      ced-10      chd-3
      dbl-1 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 unc-129
      vab-7 end
```

```
DD1   AB.plppappap      MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DD2   AB.prppappap      MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DD3   AB.plppapppa      MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DD4   AB.prppapppa      MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DD5   AB.plppappppp     MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DD6   AB.prppappppp     MN    VC    x     cav-1 cdc-42      ced-10      ced-
10    chd-3 fbxb-103    ina-1 lin-35      lin-53      npr-1 ptp-3 set-2 unc-
25    unc-25      unc-47      unc-47      unc-47      end

DVA   AB.prpppppapp     IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 dbl-1 dbl-1 fax-1 fbxb-103    ggr-2 glr-4 glr-5 gpa-14      hlh-2
      ina-1 lin-11      lin-35      lin-53      nmr-1 ptp-3 ser-2 ser-4 set-2
      zig-5 end
DVB   AB.plpapppaK      IN    R     x     egl-36      egl-36      egl-4 egl-5
      kal-1 lim-6 snt-1 unc-2 unc-25      unc-25      unc-25      unc-47
      unc-47      unc-47      end

DVC   AB.plapaappCaapaa IN    R     x     cam-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-14      chd-3 egl-4 fbxb-103    glr-1 hcp-3 ina-1 kal-1
      lin-11      lin-35      lin-53      ptp-3 sem-4 ser-4 set-2 unc-103
      end
FLPL  AB.plapaaapad     IN    H     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 che-2 che-3 des-2 eat-4 egl-44      egl-46      egl-46      glr-4
      ina-1 let-60      lin-35      lin-53      mec-10      mec-3 mec-3 mec-7
      mec-7 mec-7 osm-9 set-2 unc-8 unc-8 unc-86      end
FLPR  AB.prapaaapad     IN    H     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 che-2 che-3 des-2 eat-4 egl-44      egl-46      egl-46      glr-4
      ina-1 let-60      lin-35      lin-53      mec-10      mec-3 mec-3 mec-7
      mec-7 mec-7 osm-9 set-2 unc-8 unc-8 unc-86      end



HSNL  AB.plapppappa     MN    V     x     cam-1 cat-1 cav-1 cdc-42      cdh-3
      cdh-3 cdh-3 ced-10      cha-1 chd-3 clh-3 clh-3 eat-16      egl-21
      egl-3 egl-43      egl-44      egl-46      egl-5 egl-5 flt-1 flt-1 flt-1
      gar-2 ggr-2 glr-5 goa-1 gpb-2 grd-6 gsa-1 ham-2 ham-2 hbl-1 ida-1 ida-1
      ida-1 ida-1 ina-1 inx-4 jkk-1 jnk-1 kal-1 kal-1 lin-35      lin-53
```

```
      mab-23      mec-6 mig-1 mig-2 nhx-5 nid-1 nlp-15      nlp-3 rgs-2 sax-3
      sem-4 set-2 syg-1 tph-1 tph-1 unc-103     unc-14      unc-17      unc-2
      unc-40      unc-51      unc-53      unc-73      unc-76      unc-8 unc-
86    unc-86      end
HSNR  AB.prapppappa     MN    V     x     cam-1 cat-1 cav-1 cdc-42      cdh-3
      cdh-3 cdh-3 ced-10      cha-1 chd-3 clh-3 clh-3 eat-16      egl-21
      egl-3 egl-43      egl-44      egl-46      egl-5 egl-5 flt-1 flt-1 flt-1
      gar-2 ggr-2 glr-5 goa-1 gpb-2 grd-6 gsa-1 ham-2 ham-2 hbl-1 ida-1 ida-1
      ida-1 ida-1 ina-1 inx-4 jkk-1 jnk-1 kal-1 kal-1 lin-35      lin-53
      mab-23      mec-6 mig-1 mig-2 nhx-5 nid-1 nlp-15      nlp-3 rgs-2 sax-3
      sem-4 set-2 syg-1 tph-1 tph-1 unc-103     unc-14      unc-17      unc-2
      unc-40      unc-51      unc-53      unc-73      unc-76      unc-8 unc-
86    unc-86      end
IL1DL AB.alapappaaa     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      ina-1 lin-35      lin-53      oig-1 osm-6 sax-7 set-2 end



IL1DR AB.alappppaaa     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      ina-1 lin-35      lin-53      oig-1 osm-6 sax-7 set-2 end



IL1L  AB.alapaappaa     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 daf-9 eat-4 goa-1 ina-1 lin-35      lin-53      mec-6 oig-1 osm-6
      sax-7 set-2 end



IL1R  AB.alaappppaa     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 daf-9 eat-4 goa-1 ina-1 lin-35      lin-53      mec-6 oig-1 osm-6
      sax-7 set-2 end



IL1VL AB.alppapppaa     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 ina-1 lin-35      lin-53      oig-1 osm-6 sax-7 set-2 end



IL1VR AB.arappppaa      IN    L     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 ina-1 lin-35      lin-53      oig-1 osm-6 sax-7 set-2 end
```

```
IL2DL AB.alapappap      IN    L     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      oig-1 osm-3 osm-6 ptp-3 sax-7
      set-2 unc-5 unc-86      end


IL2DR AB.alappppap      IN    L     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      oig-1 osm-3 osm-6 ptp-3 sax-7
      set-2 unc-5 unc-86      end


IL2L  AB.alapaappp      IN    L     x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 fbxb-103    ina-1 lin-35      lin-53      nlp-11      nlp-6 npr-1
      odr-2 oig-1 osm-3 osm-6 ptp-3 sax-7 set-2 unc-5 unc-86      end


IL2R  AB.alaappppp      IN    L     x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 fbxb-103    ina-1 lin-35      lin-53      nlp-11      nlp-6 npr-1
      odr-2 oig-1 osm-3 osm-6 ptp-3 sax-7 set-2 unc-5 unc-86      end


IL2VL AB.alppappppp     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      oig-1 osm-3 osm-6 ptp-3 sax-7
      set-2 unc-5 unc-86      end


IL2VR AB.arappppppp     IN    L     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      oig-1 osm-3 osm-6 ptp-3 sax-7
      set-2 unc-5 unc-86      end


LUAL  AB.plpppaapap     IN    VC    x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 daf-5 eat-4 glr-5 gpa-10      ina-1 itr-1 lin-35      lin-53
      nlp-13      ser-2 set-2 sgk-1 end


LUAR  AB.prpppaapap     IN    VC    x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 daf-5 eat-4 glr-5 gpa-10      ina-1 itr-1 lin-35      lin-53
```

```
      nlp-13      ser-2 set-2 sgk-1 end


OLLL  AB.alpppapaa     MN    L     x     ace-1 bbs-5 cav-1 cdc-42      ced-
10    chd-3 che-13      che-13      che-2 eat-4 goa-1 ina-1 ins-11      ins-2
      ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-35
      lin-53      oig-1 osm-5 osm-5 osm-5 osm-6 sax-7 ser-2 set-2 unc-5 end


OLLR  AB.praaapapaa     MN    L     x     ace-1 bbs-5 cav-1 cdc-42      ced-
10    chd-3 che-13      che-13      che-2 eat-4 goa-1 ina-1 ins-11      ins-2
      ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-35
      lin-53      oig-1 osm-5 osm-5 osm-5 osm-6 sax-7 ser-2 set-2 unc-5 end


OLQDL AB.alapapapaa     MN    L     x     bbs-5 cav-1 cdc-42      ced-10
      chd-3 che-13      che-13      che-2 eat-20      eat-4 goa-1 ina-1 ins-
11    ins-2 ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-
35    lin-53      npr-1 ocr-4 oig-1 osm-5 osm-5 osm-5 osm-6 osm-9 osm-9 osm-9
      osm-9 sax-3 sax-7 set-2 unc-5 unc-5 end

OLQDR AB.alapppapaa     MN    L     x     bbs-5 cav-1 cdc-42      ced-10
      chd-3 che-13      che-13      che-2 eat-20      eat-4 goa-1 ina-1 ins-
11    ins-2 ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-
35    lin-53      npr-1 ocr-4 oig-1 osm-5 osm-5 osm-5 osm-6 osm-9 osm-9 osm-9
      osm-9 sax-3 sax-7 set-2 unc-5 unc-5 end

OLQVL AB.plpaaappaa     MN    L     x     bbs-5 cav-1 cdc-42      ced-10
      chd-3 che-13      che-13      che-2 eat-20      eat-4 goa-1 ina-1 ins-
11    ins-2 ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-
35    lin-53      npr-1 ocr-4 oig-1 osm-5 osm-5 osm-5 osm-6 osm-9 osm-9 osm-9
      osm-9 sax-3 sax-7 set-2 unc-5 unc-5 end

OLQVR AB.prpaaappaa     MN    L     x     bbs-5 cav-1 cdc-42      ced-10
      chd-3 che-13      che-13      che-2 eat-20      eat-4 goa-1 ina-1 ins-
11    ins-2 ins-22      ins-23      ins-3 ins-4 ins-5 ins-6 ins-7 ins-8 lin-
35    lin-53      npr-1 ocr-4 oig-1 osm-5 osm-5 osm-5 osm-6 osm-9 osm-9 osm-9
      osm-9 sax-3 sax-7 set-2 unc-5 unc-5 end

PDA   AB.prpppaaaa      MN    DC    x     ace-2 ace-3 dop-2 egl-5 itr-1 itr-1
      kvs-1 unc-8 end


PDB   ABplapappa12apa   MN    DC    EC    dbl-1 kal-1 kal-1 unc-8 unc-8 end


PDEL  ABplapapaaVLpaaa  SN    D     x     cat-2 cat-2 che-2 che-3 dat-1 dop-2
      egl-36      gpa-16      ida-1 ida-1 jkk-1 jnk-1 osm-6 pkc-1 unc-32
      unc-40      unc-53      unc-73      end
```

```
PDER  ABprapapaaVRpaaa  SN    D     x     cat-2 cat-2 che-2 che-3 dat-1 dop-2
      egl-36      gpa-16      ida-1 ida-1 jkk-1 jnk-1 osm-6 pkc-1 unc-32
      unc-40      unc-53      unc-73      end


PHAL  AB.plpppaapp      SN    CH    PH    bra-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-14      ceh-23      chd-3 che-2 che-3 daf-4 egl-43
      fbxb-103    gcy-12      goa-1 gpa-1 gpa-1 gpa-13      gpa-14      gpa-
15    gpa-2 gpa-2 gpa-3 ida-1 ida-1 ina-1 lin-11      lin-35      lin-53
      ncs-1 nlp-14      nlp-14      nlp-7 nlp-7 npr-1 ocr-2 osm-10      osm-
10    osm-10      osm-3 osm-6 osm-9 pkc-1 ptp-3 set-2 sgk-1 srb-6 srg-13
      srp-2 tax-6 tax-6 tax-6 unc-103     unc-40      end
PHAR  AB.prpppaapp      SN    CH    PH    bra-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-14      ceh-23      chd-3 che-2 che-3 daf-4 egl-43
      fbxb-103    gcy-12      goa-1 gpa-1 gpa-1 gpa-13      gpa-14      gpa-
15    gpa-2 gpa-2 gpa-3 ida-1 ida-1 ina-1 lin-11      lin-35      lin-53
      ncs-1 nlp-14      nlp-14      nlp-7 nlp-7 npr-1 ocr-2 osm-10      osm-
10    osm-10      osm-3 osm-6 osm-9 pkc-1 ptp-3 set-2 sgk-1 srb-6 srg-13
      srp-2 tax-6 tax-6 tax-6 unc-103     unc-40      end
PHBL  AB.plapppappp     SN    CH    PH    bra-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-14      ceh-23      chd-3 che-2 che-3 cog-1 daf-4 gpa-1
      gpa-1 gpa-13      gpa-14      gpa-15      gpa-2 gpa-2 gpa-3 gpa-6 gpa-9
      gpc-1 ida-1 ida-1 ina-1 lin-35      lin-53      ncs-1 ncs-1 nlp-1 nlp-1
      npr-1 ocr-2 osm-10      osm-10      osm-10      osm-3 osm-6 osm-9 pkc-1
      set-2 sgk-1 srb-6 srp-2 tax-6 tax-6 tax-6 unc-40      vab-15      end

PHBR  AB.prapppappp     SN    CH    PH    bra-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-23      chd-3 che-2 che-3 cog-1 daf-4 gpa-1 gpa-1 gpa-
13    gpa-14      gpa-15      gpa-2 gpa-2 gpa-3 gpa-6 gpa-9 gpc-1 ida-1 ida-1
      ina-1 lin-35      lin-53      ncs-1 ncs-1 nlp-1 nlp-1 npr-1 ocr-2 osm-
10    osm-10      osm-10      osm-3 osm-6 osm-9 pkc-1 set-2 sgk-1 srb-6 srp-2
      tax-6 tax-6 tax-6 unc-40      vab-15      end
PHCL  ABplappppTLpppaa  SN    M     x     ahr-1 ceh-14      ceh-14      dop-1
      ida-1 ida-1 ida-1 ida-1 mab-23      unc-73      unc-86      end


PHCR  ABprappppTRpppaa  SN    M     x     ahr-1 ceh-14      ceh-14      dop-1
      ida-1 ida-1 ida-1 ida-1 mab-23      unc-73      unc-86      end


PLML  AB.plapappppaa    SN    T     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-1 deg-3 dop-1 eat-4 eat-4 egl-46      egl-5 egl-5 gcy-35
      glr-8 gpa-16      ina-1 lin-35      lin-53      mab-21      mec-2 mec-3
      mec-3 mec-4 mec-6 mec-7 mec-7 mec-7 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1
      ptl-1 rhgf-1      sax-7 set-2 tba-1 tba-1 tba-1 tba-2 tol-1 unc-32
      unc-73      unc-86      unc-97      end
```

```
PLMR  AB.prapappppaa    SN    T     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-1 deg-3 dop-1 eat-4 eat-4 egl-46      egl-5 egl-5 gcy-35
      glr-8 gpa-16      ina-1 lin-35      lin-53      mab-21      mec-2 mec-3
      mec-3 mec-4 mec-6 mec-7 mec-7 mec-7 mps-1 mtd-1 nid-1 pag-3 pat-4 pkc-1
      ptl-1 rhgf-1      sax-7 set-2 tba-1 tba-1 tba-1 tba-2 tol-1 unc-32
      unc-73      unc-86      unc-97      end
PLNL  ABplappppTLpppap  IN    PL    x     ahr-1 cha-1 daf-1 dop-1 egl-2 gcy-
35    gcy-35      lad-2 odr-2 unc-17      unc-32      unc-53      unc-86
      end


PLNR  ABprappppTRpppap  IN    PL    x     ahr-1 cha-1 daf-1 dop-1 egl-2 gcy-
35    gcy-35      lad-2 odr-2 unc-17      unc-32      unc-53      unc-86
      end


PQR   ABplapapaaQLap    IN    EC    x     ace-3 ahr-1 ceh-13      che-2 che-3
      cki-1 daf-28      egl-2 egl-2 egl-4 gcy-32      gcy-34      gcy-35
      gcy-35      gcy-36      gcy-36      gcy-36      gcy-37      gpa-8 mab-
21    npr-1 osm-6 tax-2 tax-2 unc-86      end


PVCL  AB.plpppaapaa     IN    VC    x     ace-2 cam-1 cav-1 cdc-42      ced-
10    cfi-1 chd-3 deg-1 deg-3 des-2 egl-3 egl-36      egl-5 glr-1 glr-1 glr-1
      glr-2 glr-5 goa-1 gpa-16      ina-1 lin-35      lin-53      mab-21
      mab-9 mec-6 nhr-83      nmr-1 nmr-1 nmr-1 nmr-2 rig-1 sax-3 ser-2 set-2
      unc-8 unc-8 vab-15      end

PVCR  AB.prpppaapaa     IN    VC    x     ace-2 cam-1 cav-1 cdc-42      ced-
10    cfi-1 chd-3 deg-1 deg-3 des-2 egl-3 egl-36      egl-5 glr-1 glr-1 glr-1
      glr-2 glr-5 goa-1 gpa-16      ina-1 lin-35      lin-53      mab-21
      mab-9 mec-6 nhr-83      nmr-1 nmr-1 nmr-1 nmr-2 rig-1 sax-3 ser-2 set-2
      unc-8 unc-8 vab-15      end

PVDL  ABplapapaaVLpaapa IN    EC    x     deg-3 des-2 eat-4 egl-46      egl-
46    glr-4 goa-1 jkk-1 jnk-1 mec-10      mec-3 mec-3 mec-6 mec-7 mec-7 mec-9
      nlp-11      osm-9 pkc-1 ser-2 unc-32      unc-40      unc-86      end


PVDR  ABprapapaaVRpaapa IN    EC    x     deg-3 des-2 eat-4 egl-46      egl-
46    glr-4 goa-1 jkk-1 jnk-1 mec-10      mec-3 mec-3 mec-6 mec-7 mec-7 mec-9
      nlp-11      osm-9 pkc-1 ser-2 unc-32      unc-40      unc-86      end


PVM   ABplapapaaQLpaa   SN    T     x     ahr-1 cam-1 egl-21      egl-3 egl-
46    flt-1 gar-1 goa-1 kal-1 mec-10      mec-2 mec-3 mec-3 mec-4 mec-6 mec-7
      mec-7 mec-7 mec-7 mtd-1 pag-3 pat-4 pkc-1 sax-7 scc-1 tba-1 tba-1 tba-1
      tol-1 unc-32      unc-40      unc-53      unc-8 unc-8 unc-86      unc-
97    end
PVNL  ABplappppTLappp   IN    VC    x     ceh-14      ceh-14      unc-73
      end
```

```
PVNR  ABprappppTRappp   IN    VC    x     ceh-14      ceh-14      unc-73
      end


PVPL  AB.plppppaaa      IN    VC    x     cav-1 cdc-42      ced-10      chd-3
      fax-1 fbxb-103    ida-1 ida-1 ina-1 lin-11      lin-35      lin-53
      odr-2 ptp-3 set-2 unc-103     unc-30      unc-53      end


PVPR  AB.prpppaaa       IN    VC    R     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ida-1 ida-1 ina-1 lin-11      lin-35      lin-53      odr-2
      ptp-3 set-2 unc-103     unc-30      unc-53      end


PVQL  AB.plapppaaa      IN    VC    R     ace-2 cam-1 cav-1 cdc-42      ced-
10    ceh-14      ceh-43      chd-3 dop-1 fbxb-103    ggr-1 glr-1 glr-1 glr-1
      glr-5 gpa-14      gpa-9 hlh-14      ina-1 lin-11      lin-11      lin-
35    lin-53      nhr-83      ptp-3 sax-3 set-2 sra-6 unc-53      unc-53
      unc-6 zig-5 end

PVQR  AB.prapppaaa      IN    VC    R     ace-2 cam-1 cav-1 cdc-42      ced-
10    ceh-14      ceh-43      chd-3 dop-1 fbxb-103    ggr-1 glr-1 glr-1 glr-1
      glr-5 gpa-14      gpa-9 hlh-14      ina-1 lin-11      lin-11      lin-
35    lin-53      nhr-83      ptp-3 sax-3 set-2 sra-6 unc-53      unc-53
      unc-6 zig-5 end
PVR   AB.plapaappCaappa IN    VC    R     cav-1 cdc-42      ced-10      ceh-
14    ceh-14      chd-3 eat-4 fbxb-103    ggr-1 hcp-3 ina-1 lin-35      lin-
53    mec-12      nhr-83      ptp-3 set-2 unc-86      end

PVT   AB.plpappppa      IN    VC    R     cam-1 cav-1 cdc-42      ced-10
      ceh-14      ceh-14      chd-3 daf-1 daf-4 fax-1 fbxb-103    gpa-2 gpa-2
      gpa-3 igcm-2      ina-1 lin-35      lin-53      nhr-22      oig-3 ptp-3
      rig-3 ser-2 ser-4 set-2 tba-2 unc-42      unc-6 zig-1 zig-2 zig-3 zig-4
      zig-5 zig-8 end
PVWL  ABplappppTLppa    IN    VC    R     ahr-1 ceh-14      ceh-14      kal-1
      end


PVWR  ABprappppTRppa    IN    VC    R     ahr-1 ceh-14      ceh-14      kal-1
      end


RIAL  AB.alapaapaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      dop-2 egl-36      fbxb-103    glr-2 glr-3 glr-6 gpa-14      ina-1 lin-
35    lin-53      ptp-3 ser-2 set-2 tax-6 unc-5 unc-5 end

RIAR  AB.alaapppaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      dop-2 egl-36      fbxb-103    glr-2 glr-3 glr-6 gpa-14      ina-1 lin-
```

```
35    lin-53      ptp-3 ser-2 set-2 tax-6 unc-5 unc-5 end

RIBL  AB.plpaappap      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      dop-1 egl-4 fbxb-103    glr-4 ina-1 lin-35      lin-53      mgl-1 mgl-2
      ptp-3 ser-4 set-2 tax-6 zig-5 end

RIBR  AB.prpaappap      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      dop-1 egl-4 fbxb-103    glr-4 ina-1 lin-35      lin-53      mgl-1 mgl-2
      ptp-3 ser-4 set-2 tax-6 zig-5 end

RICL  AB.plppaaaapp     IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 fax-1 glr-5 hlh-2 ina-1 lin-11      lin-11      lin-35      lin-
53    ser-2 set-2 unc-40      end

RICR  AB.prppaaaapp     IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 fax-1 glr-5 hlh-2 ina-1 lin-11      lin-11      lin-35      lin-
53    ser-2 set-2 unc-40      end

RID   AB.alappaapa      IN    R     DC    cav-1 cdc-42      ced-10      ceh-
10    chd-3 dop-2 fbxb-103    goa-1 ina-1 kal-1 lim-4 lin-35      lin-53
      ptp-3 ser-2 set-2 unc-30      unc-53      unc-53      zig-5 end

RIFL  AB.plppapaaap     IN    R     x     cav-1 cdc-42      ced-10      chd-3
      glr-4 glr-5 ina-1 lin-35      lin-53      odr-2 set-2 unc-6 end


RIFR  AB.prppapaaap     IN    R     x     cav-1 cdc-42      ced-10      chd-3
      glr-4 glr-5 ina-1 lin-35      lin-53      odr-2 set-2 end


RIGL  AB.plppappaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      egl-3 fbxb-103    flp-18      glr-1 glr-2 ina-1 lin-35      lin-53
      nhr-83      npr-1 odr-2 ptp-3 set-2 end

RIGR  AB.prppappaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      egl-3 fbxb-103    flp-18      glr-1 glr-2 ina-1 lin-35      lin-53
      nhr-83      npr-1 odr-2 ptp-3 set-2 end

RIH   AB.prpappaaa      IN    R     x     cat-1 cav-1 cdc-42      ced-10
      chd-3 fbxb-103    ina-1 lin-35      lin-53      ptp-3 set-2 slt-1 tph-1
      unc-30      unc-5 unc-86      end

RIML  AB.plppaapap      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      cha-1 chd-3 dop-1 fbxb-103    flp-18      glr-1 glr-1 glr-4 glr-5 goa-1
      ina-1 lin-35      lin-53      nmr-1 nmr-2 ptp-3 set-2 tax-6 unc-17
      unc-40      end
RIMR  AB.prppaapap      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      cha-1 chd-3 dop-1 fbxb-103    flp-18      glr-1 glr-1 glr-4 glr-5 goa-1
      ina-1 lin-35      lin-53      nmr-1 nmr-2 ptp-3 set-2 tax-6 unc-17
      unc-40      end
RIPL  AB.alpapaaaa      IN    R     P     ahr-1 cav-1 cdc-42      ced-10
      chd-3 fbxb-103    gpa-16      ina-1 lin-35      lin-53      ptp-3 set-2
```

```
      unc-86      end

RIPR  AB.prappaaaa      IN    R     P     ahr-1 cav-1 cdc-42      ced-10
      chd-3 fbxb-103    gpa-16      ina-1 lin-35      lin-53      ptp-3 set-2
      unc-86      end

RIR   AB.prpapppaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-2 ina-1 lin-35      lin-53      ptp-3 set-2 unc-86
      end
RIS   AB.prpappapa      IN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 dop-1 fbxb-103    glr-1 ina-1 lim-6 lin-35      lin-53      ptp-3
      ser-4 set-2 tax-6 unc-25      unc-25      unc-25      unc-47      unc-
47    unc-47      end
RIVL  AB.plpaapaaa      IN    R     x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-14      fbxb-103    ina-1 lim-4 lin-35      lin-53      npr-1
      odr-2 ptp-3 set-2 unc-42      zig-5 end
RIVR  AB.prpaapaaa      IN    R     x     cam-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 daf-14      fbxb-103    ina-1 lim-4 lin-35      lin-53      npr-1
      odr-2 ptp-3 set-2 unc-42      zig-5 end
RMDDL AB.alpapapaa      MN    R     x     cav-1 cdc-42      ced-10      ceh-6
      ceh-6 chd-3 egl-4 fbxb-103    glr-2 glr-4 ina-1 lim-4 lin-35      lin-
53    mgl-1 mgl-2 ptp-3 rig-5 set-2 end
RMDDR AB.arappapaa      MN    R     x     cav-1 cdc-42      ced-10      ceh-6
      ceh-6 chd-3 egl-4 fbxb-103    glr-2 glr-4 ina-1 lim-4 lin-35      lin-
53    mgl-1 mgl-2 ptp-3 rig-5 set-2 end
RMDL  AB.alpppapad      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 egl-4 fbxb-103    glr-1 glr-4 glr-5 goa-1 ina-1 lim-4 lin-35
      lin-53      mgl-1 mgl-2 ptp-3 qui-1 rig-5 sax-3 set-2 unc-42      end
RMDR  AB.praaaapad      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      chd-3 egl-4 fbxb-103    glr-1 glr-4 glr-5 goa-1 ina-1 lim-4 lin-35
      lin-53      mgl-1 mgl-2 ptp-3 qui-1 rig-5 sax-3 set-2 unc-42      end
RMDVL AB.alppapaaa      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 csk-1 egl-4 fbxb-103    glr-2 glr-4 ina-1 lin-35
      lin-53      mgl-1 mgl-2 ptp-3 qui-1 rig-5 set-2 src-1 tax-6 tax-6 end
RMDVR AB.arapppaaa      MN    R     x     cam-1 cav-1 cdc-42      ced-10
      ceh-6 ceh-6 chd-3 csk-1 egl-4 fbxb-103    glr-2 glr-4 ina-1 lin-35
      lin-53      mgl-1 mgl-2 ptp-3 qui-1 rig-5 set-2 src-1 tax-6 tax-6 end
RMED  AB.alapppaap      MN    R     x     avr-15      cam-1 cav-1 cdc-42
      ced-10      ceh-10      ceh-17      chd-3 fbxb-103    ina-1 lim-4 lim-4
      lim-6 lin-35      lin-53      odr-2 ptp-3 set-2 slt-1 unc-25      unc-
47    end
RMEL  AB.alaaaarlp      MN    R     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 clh-6 clh-6 fbxb-103    glr-1 ina-1 lim-4 lim-6 lin-35      lin-
53    odr-2 ptp-3 ser-2 set-2 unc-25      unc-47      end
RMER  AB.alaaaarrp      MN    R     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 clh-6 clh-6 fbxb-103    glr-1 ina-1 lim-4 lim-6 lin-35      lin-
53    odr-2 ptp-3 ser-2 set-2 unc-25      unc-47      end
RMEV  AB.plpappaaa      MN    R     x     avr-15      cam-1 cav-1 cdc-42
      ced-10      chd-3 fbxb-103    ina-1 lim-4 lim-4 lim-6 lin-35      lin-
53    odr-2 ptp-3 set-2 unc-25      unc-47      end
RMFL  ABplapaapG2al     MN    R     x     end
```

```
RMFR  ABplapaapG2ar     MN    R     x     end


RMGL  AB.plapaaapp      IN    R     x     avr-15      cam-1 cam-1 cav-1 cdc-
42    ced-10      chd-3 fbxb-103    glr-4 glr-5 goa-1 ina-1 lin-35      lin-
53    ncs-1 ncs-1 npr-1 ptp-3 sax-3 sax-7 set-2 unc-86      end
RMGR  AB.prapaaapp      IN    R     x     avr-15      cam-1 cam-1 cav-1 cdc-
42    ced-10      chd-3 fbxb-103    glr-4 glr-5 goa-1 ina-1 lin-35      lin-
53    ncs-1 ncs-1 npr-1 ptp-3 sax-3 sax-7 set-2 unc-86      end
RMHL  ABprpaaaapG1l     MN    R     x     end


RMHR  ABprpaaaapG1r     MN    R     x     end


SAADL AB.alppapapa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-4 ina-1 lad-2 lim-4 lin-35      lin-53      npr-1 ptp-3
      set-2 unc-42      end
SAADR AB.arapppapa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-4 ina-1 lad-2 lim-4 lin-35      lin-53      npr-1 ptp-3
      set-2 unc-42      end
SAAVL AB.plpaaaaaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-4 ina-1 lad-2 lim-4 lin-35      lin-53      ptp-3 set-2
      unc-42      end
SAAVR AB.prpaaaaaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-4 ina-1 lad-2 lim-4 lin-35      lin-53      ptp-3 set-2
      unc-42      end
SABD  AB.plppapaap      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fax-1 fbxb-103    glr-4 glr-5 ina-1 lin-35      lin-53      ptp-3 ser-2
      set-2 sng-1 unc-8 end
SABVL AB.plppapaaaa     IN    R     x     cav-1 cdc-42      ced-10      ceh-6
      ceh-6 chd-3 del-1 fax-1 glr-5 ina-1 lin-35      lin-53      ser-2 set-2
      sng-1 end
SABVR AB.prppapaaaa     IN    R     x     cav-1 cdc-42      ced-10      ceh-6
      ceh-6 chd-3 del-1 fax-1 glr-5 ina-1 lin-35      lin-53      ser-2 set-2
      sng-1 end
SDQL  ABplapapaaQLpap   IN    R     x     ahr-1 cam-1 cha-1 egl-21      egl-3
      gcy-35      gcy-35      jkk-1 jnk-1 lad-2 npr-1 pkc-1 scc-1 ser-2 unc-
17    unc-32      unc-86      end
SDQR  ABprapapaaQRpap   IN    R     x     ahr-1 cam-1 cha-1 egl-21      gcy-
35    gcy-35      jkk-1 jnk-1 lad-2 npr-1 pkc-1 scc-1 ser-2 unc-17      unc-2
      unc-32      unc-86      end
SIADL AB.plpapaapa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    chd-3 cog-1 dop-2 fbxb-103    ggr-2 goa-1 ina-1 lim-4 lim-4 lin-35
      lin-53      ptp-3 sax-3 ser-2 set-2 sup-9 unc-103     unc-93      end
SIADR AB.prpapaapa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    chd-3 cog-1 dop-2 fbxb-103    ggr-2 goa-1 ina-1 lim-4 lim-4 lin-35
      lin-53      ptp-3 sax-3 ser-2 set-2 sup-9 unc-93      end
SIAVL AB.plpapappa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    chd-3 cog-1 dop-2 fbxb-103    ggr-2 goa-1 ina-1 lim-4 lim-4 lin-35
      lin-53      odr-2 ptp-3 sax-3 ser-2 set-2 sup-9 unc-93      end
SIAVR AB.prpapappa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    chd-3 cog-1 dop-2 fbxb-103    ggr-2 goa-1 ina-1 lim-4 lim-4 lin-35
```

```
      lin-53      odr-2 ptp-3 sax-3 ser-2 set-2 sup-9 unc-103     unc-93
      end
SIBDL AB.plppaaaaa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
24    chd-3 dop-2 fbxb-103    glr-4 ina-1 lin-35      lin-53      ptp-3 sax-3
      set-2 unc-42      end
SIBDR AB.prppaaaaa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
24    chd-3 dop-2 fbxb-103    glr-4 ina-1 lin-35      lin-53      ptp-3 sax-3
      set-2 unc-42      end
SIBVL AB.plpapaapp      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    ceh-24      chd-3 dop-2 fbxb-103    glr-4 ina-1 lin-35      lin-53
      ptp-3 sax-3 set-2 unc-42      end
SIBVR AB.prpapaapp      IN    R     x     cav-1 cdc-42      ced-10      ceh-
17    ceh-24      chd-3 dop-2 fbxb-103    glr-4 ina-1 lin-35      lin-53
      ptp-3 sax-3 set-2 unc-42      end
SMBDL AB.alpapapapp     IN    R     x     cav-1 cdc-42      ced-10      chd-3
      cog-1 ina-1 lin-35      lin-53      npr-1 odr-2 set-2 unc-42      end

SMBDR AB.arappapapp     IN    R     x     cav-1 cdc-42      ced-10      chd-3
      cog-1 ina-1 lin-35      lin-53      npr-1 odr-2 set-2 unc-42      end

SMBVL AB.alpapappp      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      odr-2 ptp-3 set-2 unc-42
      end
SMBVR AB.arappappp      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    ina-1 lin-35      lin-53      odr-2 ptp-3 set-2 unc-42
      end
SMDDL AB.plpapaaaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      egl-4 fbxb-103    ggr-2 ggr-2 glr-1 glr-4 goa-1 ina-1 lad-2 lim-4 lin-
35    lin-53      mgl-2 odr-2 ptp-3 sax-3 set-2 unc-103     wrk-1 end
SMDDR AB.prpapaaaa      IN    R     x     cav-1 cdc-42      ced-10      chd-3
      egl-4 fbxb-103    ggr-2 ggr-2 glr-1 glr-4 goa-1 ina-1 lad-2 lim-4 lin-
35    lin-53      mgl-2 odr-2 ptp-3 sax-3 set-2 unc-103     wrk-1 end
SMDVL AB.alppappaa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
24    chd-3 egl-4 fbxb-103    ggr-1 ggr-2 glr-1 glr-4 goa-1 ina-1 lad-2 lim-4
      lin-35      lin-53      mgl-2 odr-2 ptp-3 sax-3 set-2 wrk-1 end

SMDVR AB.arappppaa      IN    R     x     cav-1 cdc-42      ced-10      ceh-
24    chd-3 egl-4 fbxb-103    ggr-1 ggr-2 glr-1 glr-4 goa-1 ina-1 lad-2 lim-4
      lin-35      lin-53      mgl-2 odr-2 ptp-3 sax-3 set-2 wrk-1 end

URADL AB.plaaaaaaa      MN    R     x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 daf-9 daf-9 fbxb-103    flp-21      glr-4 ina-1 lin-35      lin-
53    ptp-3 set-2 smp-2 unc-86      end

URADR AB.arpapaaaa      MN    R     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 daf-9 fbxb-103    flp-21      glr-4 ina-1 lin-35      lin-53
      ptp-3 set-2 smp-2 unc-86      end

URAVL AB.plpaaapaa      MN    R     x     cav-1 cdc-42      ced-10      chd-3
      daf-9 daf-9 daf-9 fbxb-103    flp-21      glr-4 ina-1 lin-35      lin-
53    ptp-3 set-2 smp-2 unc-86      end
```

```
URAVR AB.prpaaapaa      MN    R     x     cav-1 cdc-42      ced-10      cfi-1
      chd-3 daf-9 daf-9 daf-9 fbxb-103    flp-21      glr-4 ina-1 lin-35
      lin-53      ptp-3 set-2 smp-2 unc-86      end

URBL  AB.plaapaapa      IN    R     H     cav-1 cdc-42      ced-10      cha-1
      chd-3 fbxb-103    glr-5 glr-8 ina-1 lin-35      lin-53      ptp-3 set-2
      unc-17      unc-86      end

URBR  AB.praapaapa      IN    R     H     cav-1 cdc-42      ced-10      cha-1
      chd-3 fbxb-103    glr-5 glr-8 ina-1 lin-35      lin-53      ptp-3 set-2
      unc-17      unc-86      end

URXL  AB.plaaaaappp     IN    R     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 egl-2 egl-2 fax-1 fbxb-103    gcy-32      gcy-34      gcy-35
      gcy-35      gcy-36      gcy-36      gcy-36      gcy-37      gpa-8 ina-1
      lin-35      lin-53      npr-1 pef-1 rig-6 set-2 sra-10      tax-4 unc-
86    end
URXR  AB.arpapaappp     IN    R     x     ahr-1 cam-1 cav-1 cdc-42      ced-
10    chd-3 egl-2 egl-2 fax-1 fbxb-103    gcy-32      gcy-34      gcy-35
      gcy-35      gcy-36      gcy-36      gcy-36      gcy-37      gpa-8 ina-1
      lin-35      lin-53      npr-1 pef-1 rig-6 set-2 sra-10      tax-4 unc-
86    end
URYDL AB.alapapapp      IN    R     H     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-1 glr-4 ina-1 lin-35      lin-53      ptp-3 set-2 tol-1
      unc-86      end

URYDR AB.alapppapp      IN    R     H     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-1 glr-4 ina-1 lin-35      lin-53      ptp-3 set-2 tol-1
      unc-86      end

URYVL AB.plpaaappp      IN    R     H     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-1 glr-4 ina-1 lin-35      lin-53      ptp-3 set-2 tol-1
      unc-86      end

URYVR AB.prpaaappp      IN    R     H     cav-1 cdc-42      ced-10      chd-3
      fbxb-103    glr-1 glr-4 ina-1 lin-35      lin-53      ptp-3 set-2 tol-1
      unc-86      end

VA1   AB.prapaapWpa     MN    VC    x     del-1 unc-4 end


VA2   ABplapaappP2aaaa  MN    VC    x     dbl-1 exc-7 pag-3 unc-6 end


VA3   ABplappaaaP3aaaa  MN    VC    x     dbl-1 exc-7 mab-21      pag-3 unc-6
      end
VA4   ABplappaaaP4aaaa  MN    VC    x     dbl-1 exc-7 pag-3 unc-6 end
VA5   ABplappaapP5aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
VA6   ABplappaapP6aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
VA7   ABplappappP7aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
VA8   ABplappappP8aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
VA9   ABplapapapP9aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
VA10  ABplapapap10aaaa  MN    VC    x     dbl-1 pag-3 unc-6 end
```

```
VA11  ABplapappa11aaaa  MN    VC    x     dbl-1 lin-29      pag-3 unc-42
      unc-6 end
VA12  ABplapappa12aaaa  MN    VC    x     avr-15      dbl-1 pag-3 unc-6 end
VB1   ABplapaappP1aaap  MN    VC    x     dbl-1 del-1 pag-3 end
VB2   AB.prapaappWaap   MN    VC    x     dbl-1 del-1 pag-3 end
VB3   ABplapaappP2aaap  MN    VC    x     dbl-1 exc-7 pag-3 unc-6 end
VB4   ABplappaaaP3aaap  MN    VC    x     dbl-1 exc-7 pag-3 unc-6 end
VB5   ABplappaaaP4aaap  MN    VC    x     dbl-1 exc-7 pag-3 unc-6 end
VB6   ABplappaapP5aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VB7   ABplappaapP6aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VB8   ABplappappP7aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VB9   ABplappappP8aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VB10  ABplapapapP9aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VB11  ABplapapap10aaap  MN    VC    x     dbl-1 pag-3 unc-6 end
VC1   ABplappaaaP3aap   MN    VC    V     ida-1 unc-4 vab-7 end
VC2   ABplappaaaP4aap   MN    VC    V     ida-1 vab-7 end
VC3   ABplappaapP5aap   MN    VC    V     ida-1 vab-7 end
VC4   ABplappaapP6aap   MN    VC    V     cat-1 ida-1 end
VC5   ABplappappP7aap   MN    VC    V     cat-1 ida-1 end
VC6   ABplappappP8aap   MN    VC    V     ida-1 end
VD1   AB.prapaapWpp     MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD2   ABplapaappP1app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD3   ABplapaappP2app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD4   ABplappaaaP3app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD5   ABplappaaaP4app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD6   ABplappaapP5app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD7   ABplappaapP6app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD8   ABplappappP7app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD9   ABplappappP8app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD10  ABplapapapP9app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD11  ABplapapap10app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
VD12  ABplapappa11app   MN    VC    x     ced-10      npr-1 unc-25      unc-
42    unc-47      end
VD13  ABplapappa12app   MN    VC    x     ced-10      npr-1 unc-25      unc-
47    end
2199
ADAL  AIBL  1
ADAL  AIBR  1
ADAL  AVAR  1
ADAL  AVBL  1
ADAL  AVBR  1
ADAL  AVDL  1
```

```
ADAL  AVEL  1
ADAL  AVJR  1
ADAL  FLPR  1
ADAL  RICL  1
ADAL  RICR  1
ADAL  RIML  1
ADAL  RIPL  1
ADAL  SMDVR 1
ADAR  AIBL  1
ADAR  AIBR  1
ADAR  AVAL  1
ADAR  AVBL  1
ADAR  AVBR  1
ADAR  AVEL  1
ADAR  AVJL  1
ADAR  RICL  1
ADAR  RIMR  1
ADAR  RIPR  1
ADAR  RIVR  1
. . .
VD13  VA12  1
AIAR  ASGL  1
AIMR  ASGL  1
AINL  ASGL  1
AWBR  ASGL  1
AWCR  ASGL  1
CEPVR ASGL  1
DVA   SMBDR 1
DVA   SMBVL 1
DVA   SMBVR 1
PVR   URADL 1
DVA   VA2   1
DVA   VA12  1
DVA   VB1   1
```

Appendix V
User's Guide

It is assumed that you are working on a UNIX or Linux system. Your system must support a C++ and C compiler, preferably g++ and gcc. Your system must also support the C++ Standard Template Library (STL) or the programs will not work.

General Notes

1. All output files are text files. Peruse them with a text editor.
2. All input files are also text files, you may peruse them with a text editor. But in the case of input files be careful of the modifications you make, as the input file must be well formed to work with the program (See the section on 'Valid File Formats.').
3. When compiling all header files referenced in the program text must be in the same directory as the program.
4. It is assumed that you are compiling and running the programs under UNIX (of Linux). Load the program disk into a Windows system and transfer the files to directories on the UNIX system using FTP or Putty.
5. It is not advisable to try running the programs on Windows as the programs are compute intensive and the programs will be too slow or crash on a Windows system.
6. The UNIX system should have at least a 32-bit CPU, a 64-bit CPU is better, with plenty of memory and disk space.
7. The UNIX system should support the GNU g++ and gcc compilers and MUST support the C++ Standard Template Library (STL). If the Standard Template Library is not present, the programs will not work.

Compiling the programs.

If the program has an extension .cpp, use g++ to compile the program. You must be sure that all header files referenced in the program are in the same directory as the program. The command line is "g++ programname.cpp -o programname". There is one program written in C rather than C++: LinDistCL.c. Compile this program with gcc. The command line is "gcc LinDistCL.c -o lindist". In this case your executable will be callled "lindist."

Running individual programs.

Next, we will demonstrate how to invoke each of the programs. Input files are provided on the CD. You may use these for a test run. In your own research, you may modify the input files. Modifications will work with the programs ONLY if you adhere to the file format that the program expects. The specifics of the file format will be given.

Neuronpathstring

Neuronpathstring gives the path through the synaptic network from an input neuron to an output neuron. First compile the program. Use "g++ neuronpathstring.cpp -o neuronpathstring". Be

sure all header files referenced in the include statements of the program text are in the current directory. Neuronpathstring is invoked under UNIX as "./neuronpathstring graphname inputneuron outputneuron," where 'neuronpathstring' is the name of the program, 'graphname' is a valid graph file, 'inputneuron' is a string that is the name of a neuron found in the graph, typically a sensory neuron, and 'outputneuron' is a string that is the name of a neuron found in the graph, typically a motor neuron. A sample actual run is "./neuronpathstring nrsynapses.txt ASKL VD13". This should return the output: "ASKL AIAL ADAL AVAR VD13". Errors include the following. If a name other than the name of a neuron in the graph is given, the program will terminate with the message "Couldn't find vertex." If a path doesn't exist from the first neuron to the second in the graph, the program will finish but the pathlength will be 0 and there will be no path output. If not all command line inputs are given the program will terminate and indicate proper usage. If command line inputs are given in the wrong order, for example, the first parameter is not a graph file, the program will terminate with "Could not open graph file," as it will if the graph file is not in the proper format.

Neuronpaths

Neuronpaths gives the paths through the synaptic network from a set of input neurons (e.g. sensory) to a set of output neurons (e.g. motor). First compile the program. Use "g++ neuronpaths.cpp -o neuronpaths". Be sure all header files referenced in the include statements of the program text are in the current directory. The command line structure for 'neuronpaths' is: "./neuronpaths graphname inputneurons outputneurons neuronpathlist", where 'neuronpaths' is the name of the program, 'graphname' is the name of a valid graph file, 'inputneurons' is a list of input neurons, 'outputneurons' is a list of output neurons, and 'neuronpathlist' is the name of the output file where you want to send the output of 'neuronpaths.' A sample invocation would be: "./neuronpaths nrsynapses.txt fullsensory.txt fullmotor.txt neuronpathsoutput". In this case the file 'neuronpathsoutput' will contain the output of neuronpaths. If you want to capture messages including error messages given by your run, append a "> errmsgs" to the command line, and the error messages will be in the file 'errmsgs". Errors include the following. If the first parameter is not a properly formed graph file, the program will terminate with "Cannot open graph file." If either the input neuron file or output neuron file are not in the proper format, the program will terminate: proper format for each is the number of neurons on line one and the name of each neuron on each subsequent line. If the wrong number of parameters are given the program will indicate proper usage.

Neuronenv

Neuronenv outputs the neurons and synaptic network (a graph) of all neurons impinging upon a specific synaptic pathway. First compile the program. Use "g++ neuronenv.cpp -o neuronenv". Be sure all header files referenced in the include statements of the program text are in the current directory. The command line structure for 'neuronenv' is "./neuronenv graphname inputneuron outputneuron outputfile" where 'neuronenv' is the name of the program, 'graphname' is the name of a valid graph file, 'inputneuron' is a string that represents the name of a neuron in the graph (a valid vertex), which is typically a sensory neuron, and 'outputneuron' is the same but typically represents a motor neuron, and 'outputfile' is the name of the output file where you want the

results of the program to be stored. A sample invocation would be: "./neuronenv nrsynapses.txt ASKL VD13 neuronenvout". The file 'neuronenvout' would contain the results of running the program. If you want to capture the messages and error messages while running the program, append a "> errmsgs" to the command line. Errors include the following. Failure to include the proper number of parameters will result in the program telling you the proper usage. Failure to include a well formed graph file as the second parameter will result in termination with the message "Cannot open graph file." Giving a vertex that does not exist in the graph file for 'inputneuron' or 'outputneuron' will result in termination and the error message "Couldn't find vertex." If there is no path from the inputneuron to the outputneuron, the output will not be complete (it will not contain a graph). Remember that proper output from this program is a graph file itself, i.e., a graph of all neurons which impinge upon the path from inputneuron to outputneuron.

Neuronenvgenes

Neuronenvgenes gives the genes associated with neurons which impinge upon a specified synaptic pathway. First compile the program. Use "g++ neuronenvgenes.cpp -o neuronenvgenes". Be sure all header files referenced in the include statements of the program text are in the current directory. The command line structure for 'neuronenvgenes' is: "./neuronenvgenes graphname inputneuron outputneuron > outputfile" where 'neuronenvgenes' is the name of the program, 'graphname' is the name of a valid combined graph file (this will be explained in a minute), 'inputneuron' is the string name of a neuron that is a vertex in the graph, typically a sensory neuron, and 'outputneuron' is the same, but typically a motor neuron, and 'outputfile' is the name of the file where you want to put the output. A sample invocation would be: "./neuronenvgenes nrsynapsesgenes.txt ASKL VD13 > results". Please note that in this case a different graph file is used: 'nrsynapsesgenes.txt,' which is the combined file, i.e., it combines information about synapses and genes into one file. 'Results' is the file where the program results are stored. Possible errors include the following. If 'nrsynapsesgenes.txt' or a valid combined file (this will be explained below) is not the first parameter, the program will terminate. If either the input neuron or output neuron is not a neuron found in the graph, the program will terminate. If there is no path from the input neuron to the output neuron the program will return no results. If you do not append "> results" your results will not be stored for you to peruse.

Pathgenes

Pathgenes gives the genes associated with a specific synaptic pathway. First compile the program. Use "g++ pathgenes.cpp -o pathgenes". Be sure all header files referenced in the include statements of the program text are in the current directory. The command line structure for 'pathgenes' is: "./pathgenes graphfile inputneuron outputneuron > outputfile" where 'graphfile' is a valid graph file, inputneuron is the name of a neuron in the graph file, typically a sensory neuron, and 'outputneuron' is the name of another neuron in the graph file, typically a motor neuron, and 'outputfile' is where you would like to place the results of the program. A sample invocation would be: "./pathgenes nrsynapsesgenes.txt ASKL VD13 > pathgenesresults". Note again that the combined synapse and gene file is used. The output of pathgenes is placed in the file 'pathgenesresults". Errors will occur in the following cases. If the

combined file is not used as the first parameter the program will fail. Both the input neuron and the output neuron must be vertices in the combined graph file. If you do not use "> outputfilename" your results will not be stored in a file.

Pathsgenes

Pathsgenes gives the genes associated with a set of synaptic pathways that occur between an input (sensory) and output (motor) set of neurons. The next utility is 'pathsgenes.' Note this differs from "pathgenes" as we are talking about multiple paths, hence the 's' after 'path.' First compile the program. Use "g++ pathgenes.cpp -o pathsgenes". Be sure all header files referenced in the include statements of the program text are in the current directory. The command structure for 'pathsgenes' is: "./pathsgenes graphfile inputneurons outputneurons outputfile". Note that we are in this case inputing a set of input neurons and a set of output neurons, not one input neuron and one output neuron. Note again that we will use the combined synapse and gene file. And note that in this case the output file name is the fourth parameter. A sample invocation would be: "./pathsgenes nrsynapsesgenes.txt fullsensory.txt fullmotor.txt pathsgenesoutput". In this case the output will be stored in the file 'pathgenesoutput". Errors again will occur if you do not use the combined synapse and gene file as the first parameter, if you do not use a well-formed input neuron list (with the number of neurons on the first line and the name of each neuron on each subsequent line) or a well-formed output neuron list. You can capture error output by appending "> errmsgs" to the command line.

Subpaths

Next is our most exciting utility 'subpaths.' Subpaths gives the set of synaptic pathways that go through a set of neuronal subpaths. This program is useful in identifying ganglia and other high throughput neurons. First compile the program. Use "g++ subpaths.cpp -o subpaths". Be sure all header files referenced in the include statements of the program text are in the current directory. The command structure of 'subpaths' is: "./subpaths graphfile inputneurons outputneurons outputfilename". A sample invocation would be: "./subpaths nrsynapses.txt fullsensory.txt fullmotor.txt subpathsoutput". Note here we are using the standard graph file, not the combined synapses and genes file, the input file needs to be a well formed list of input neurons (sensory neurons), with the first line the number of neurons and the name of each neuron on each subsequent line, the same for the list of output neurons, and the output file may be called whatever you want. To capture the error messages, append "> errmsgs" to the command line. Again, errors will occur if the first parameter is not a well formed graph file (neurons and synapses only). Errors will occur if the input neuron list or output neuron list is not well formed. Note, in analyzing the subpaths output, use an editor facility like 'find' to look for specific subpaths.

Pathgenespairs

Pathgenespairs identifies genes associated with contiguous and non-contiguous neurons along a pathway. First compile the program. Use "g++ pathgenespairs.cpp -o pathgenespairs". Be sure all header files referenced in the include statements of the program text are in the current directory. The command structure of 'pathgenespairs' is: "./pathgenespairs graphfile

inputneuron outputneuron > outputfile". A sample invocation is: "./pathgenespairs nrsynapsesgenes.txt ASKL VD13 > pathgenespairsoutput". The file 'pathgenespairsoutput' will contain your results. Here the combined synapse and gene file must be used. Errors will occur if the first parameter is not the combined graph file, if the input neuron or output neuron are not valid vertices in the graph file, or if the wrong number of parameters are given. The output file is a text file, peruse it with a text editor like 'vi'.

Strongcomponents

Strongcomponents partitions the synaptic network into 'strong components.' First compile the program. Use "g++ StrongComponentsCL.cpp -o strongcomponents". Be sure all header files referenced in the include statements of the program text are in the current directory. The command structure of 'strongcomponents' is: "./strongcomponents graphfile > outputfile". A sample invocation is: "./strongcomponents nrsynapses.txt > strongcompout". The file 'strongcompout' will contain the results. Note that the input file is the synapse only file 'nrsynapses.txt". Errors will occur if the first and only parameter is not a well-formed graph file. Your results will not be saved in a file unless you append "> filename" on the command line.

Develstages

Develstages divides neural, genetic, and synaptic development into a set of development stages. First compile the program. Use "g++ develstages.cpp -o develstages". Be sure all header files referenced in the include statements of the program text are in the current directory. The command structure of 'develstages' is: "./develstages graphfile outputfile". A sample invocation is: "./develstages nrsynapsesgenes.txt develout". In this case the results will be stored in the file 'develout". Note that develstages uses the combined synapse and gene file 'nrsynapsesgenes.txt." The output is voluminous. Contained in the output file are the graph files (neuron list followed by synapse list) of each development stage, but you will have to dig through the file to extract these. The total number of synapses per development stage is given after the synapse list. To create a valid graph file, you will have to put this number immediately before the synapse list. An error will occur if the first parameter is not a well formed combined synapse and gene file. An error will occur if you do not specify an output file as the second parameter. If you wish to capture error messages, append "> errmsgs" to the end of the command line.

Syndist

The syndist program generates a matrix of synaptic distances (graph distances). First compile the program. Use: "g++ SynDistCL.cpp -o syndist". Be sure all header files referenced in the include statements of the program text are in the same directory as the SynDistCL.cpp program. The command structure of syndist is: "./syndist graphfile outputfile". A sample invocation is: "./syndist nrsynapses.txt syndistmatrix". In this case the matrix of synaptic distances is stored in the file 'syndistmatrix". Be sure the first parameter is a valid graph file (synapses only), otherwise you will get an error.

Lindist

The lindist program generates a matrix of lineage distances. First compile the program. The

program is written in C rather than C++, so you must use 'gcc' rather than 'g++'. Use: "gcc LinDistCL.c -o lindist". Be sure all header files referenced in the include statements of the program text are in the same directory as 'LinDistCL.c". The command line structure is: "./lindist inputfile outputfile". A sample invocation is "./lindist lineage280.txt lindistmatrix". Note that the first parameter is a file which contains a list of neurons followed by their lineage strings. The second parameter is a file name of choice where you would like to store the lineage distance matrix. Errors will occur if you don't use a file with the same format as 'lineage280.txt". An error will occur if you don't specify an output file.

Bitgenes

Bitgenes is a program which creates a binary representation of which genes are switched on or off in each neuron. First compile the program. Use "g++ bitgenes.cpp -o bitgenes". The command structure of 'bitgenes' is: "./bitgenes graphname > outputfile". A sample invocation is: "./bitgenes nrsynapsesgenes.txt > genebitvector". The file 'genebitvector' will contain the binary represenation of which genes are switched on or off per neuron. Note that this program uses the combined synapse and gene file 'nrsynapsesgenes.txt". Errors will occur if you don't use the combined file as the first parameter. You must append "> outputfilename" to place the bit vectors in a file.

NewEdgesTab

'NewEdgesTab' is a program which generates a set of edges based upon an input neuron set. An example would be if you wanted to generate all edges on the left side of the nervous system. In that case a file containing the left neurons would be used in combination with a file containing all edges. The file containing all edges is not provided here; you can create it by editing out the neurons from 'nrsynapses.txt' and copying this file to 'edgemaster'. Do not destroy or copy over the file 'nrsynapses.txt' as you will need this for other programs. In this case the first line in the file must be the number of edges. First compile the program. Use: "g++ NewEdgesTab.cpp -o newedges". The command line structure is: "./newedges newnodelist edgemaster newedges". If you want to capture the error messages, append "> errmsgs" to the command line. A sample invocation would be: "newedges LeftNeurons.txt edgemaster leftedges". In this case the synapses for the left side of the nervous system will be stored in 'leftedges'. Errors will occur if the parameters are not in proper order, or if the first line in the 'edgemaster' file is not the number of edges in the master file.

NewLineage

'NewLineage' is a program which generates a novel lineage string list based upon an input neuron set. An example would be if you wanted to study just the development of the left side of the organism. In this case you would use 'NewLineage' to generate a list of lineage strings for just the left side of the organism. Or perhaps you would like to study the development of a specific subsystem. First compile the program. Use: "g++ NewLineage.cpp -o newlineage". Make sure all header files referenced in the include statements of the program text are in the same directory as the program. The command line structure of 'newlineage' is: "./newlineage

newnodelist masterlineagefile newlineagefile". A sample invocation would be: "./newlineage LeftNeurons.txt Lineage280.txt leftlineage". In this case the lineage strings for the left side of the nervous system would be stored in the file 'leftlineage'. Errors will occur if the parameters are not in the proper order, or if the newnodelist does not begin with the number of nodes, or if the masterlineagefile does not begin with the number of lineage strings.

Nrsynapses

'Nrsynapses' simply generates a non-redundant list of synapses, in case your orginal synapse file contains redundancies. First compile the program. Use: "g++ nrsynapses.cpp -o nrsynapses". The 'nr' stands for 'non-redundant.' The command line structure is: "./nrsynapses ingraph outgraph". A sample invocation would be "./nrsynapses Syn2.txt newnrsynapses", where 'Syn2.txt' is a redundant synapse file. 'Syn2.txt' is not provided here, but if you have a redundant synapses file you may use this utility. The synapse file provided on the disk is already non-redundant. Errors will occur if the first parameter is not a well-formed graph file.

Lisp-programs

To use the Lisp programs, simply copy and paste them into a Lisp interpreter. Most of these programs were written to allow the programs in this research to be used with the Biobike interface at http://www.biobike.org .

Doing Your Own Research

Our examples have been based upon the files provided from our own research. To do your own research, modify the input files. In modifying the input files, the format of the file must remain the same, i.e., your input files must be in a valid format for the programs to work. As an example, you may want to use a graph that represents just a subsystem of the nervous system. You may want to submit a subset of input sensory neurons and a subset of output motor neurons that describe the inputs and outputs of a certain neural circuit. Whatever modifications you make, the valid format of the file must be preserved.

Valid File Formats

The Synapse Graph File

The synapse only graph file must have the following format. The first line must be the number of neurons. Next, is the neuron list, with the name of each neuron must appear on each subsequent line. This is followed by the exact number of synapses. This is followed by each synapse on each subsequent line, where the synapse has the form of: neuron1 neuron2 weight. An example is "ASKL AVAL 1". The weight will always be set to 1. Every neuron in every synapse must be a neuron that appears in the neuron list. If this is not the case the program will abort. Even a misspelling of a neuron name will result in the program aborting. No neuron may synapse upon itself.

The Combined Synapse-Gene File

For other programs, as described above, the combined synapse-gene file is used. The combined

synapse-gene file has the following format. The first line contains the number of neurons. Each subsequent line describes the neuron and its genes as follows. Per line, first there is the neuron named, followed by a tab, then the lineage string followed by a tab, then the main type (either 'IN', 'SN', or 'MN') followed by a tab, then a valid subtype or 'x' followed by a tab (use 'x' if you want to leave the subtype blank), then a second subtype or 'x' followed by a tab, then the gene list, i.e., the name of each gene associated with the neuron, separated by a blank, where the end of the line ends with the word 'end'. This list of neurons and their genes is followed by the number of synapses. Next is each synapse per line, in the form 'neuron1 neuron2 weight'. Note that the number of neurons must correspond to the exact number of neurons in the neuron list. Likewise, the number of synapses must correspond to the exact number of synapses in the synapse list. If this is not the case the program will abort. Again, as in the synapse graph file, if a neuron name is used in the synapse list that is not in the neuron list, the program will abort. Neurons cannot synapse upon themselves. You may want to modify the combined synapse-gene file to specify more subtypes, or to add more genes per neuron. You may want a different neuron list or a different synapse list. You can make all of those modifications for new research so long as the format is not disturbed.

Other Files

'Fullsensory.txt' and 'fullmotor.txt', as well as 'LeftNeurons.txt' and 'RightNeurons.txt' all have the format of the number of neurons on the first line followed by the list of neurons on each subsequent line. You may want to create similar files, for example, for only chemosensory neurons. 'LeftLineage.txt' and 'RightLineage.txt' may be used with lindist. These files have the number of neurons on the first line, followed by the name of the neuron and its lineage string on each subsequent line.

Appendix VI
Contents of the Compact Disk

Folder: Programs:

Bitgenes.cpp	Generates a bit representation of neuron gene sets
D_except.h	Exceptions header file for graph structure
D_galgs.h	Graph algorithms
D_graph.h	The graph data structure
D_heap.h	Heap header file for graph structure
D_pqueue.h	Priority queue header file for graph structure
D_stiter.h	Iterator header file for graph structure
D_stree.h	Tree header file for graph structure
D_util.h	Utilities header file for graph programs
D_vector.h	Vector data structure and routines
Develstages.cpp	Program that generates the developmental model
LinDistCL.c	Generates the lineage distance matrix
Lisp-programs	Lisp programs relating these programs to Biobike
Neuronenv.cpp	Finds neurons impinging upon a synaptic pathway
Neuronenvgenes.cpp	Finds genes of neurons impinging upon a synaptic pathway
Neuronpaths.cpp	Generates synaptic pathways from a set of inputs and outputs
Neuronpathstring.cpp	Finds the synaptic pathway from a sensory to motor neurons
NewEdgesTab.cpp	Generates a set of synapses based upon an input file
NewLineage.cpp	Generates lineage strings based upon an input file
Nrsynapses.cpp	Generates a non-redundant set of synapses
Pathgenes.cpp	Finds genes associated with a synaptic pathway
Pathgenepairs.cpp	Finds genes of contiguous and non-contiguous neurons
Pathsgenes.cpp	Finds genes associated with a set of synaptic pathways
StrongComponentsCL.cpp	Partitions the nervous system graph into strong components
Subpaths.cpp	Finds high throughput neurons and ganglia
SynDistCalcCL.cpp	Generates the synaptic distance matrix

Folder: InputFiles:

Fullmotor.txt	A list of all motor neurons
Fullsensory.txt	A list of all sensory neurons
LeftNeurons.txt	A list of neurons on the left side of the nervous system
LeftLineage.txt	Lineage strings on the left side of the nervous system
Lineage280.txt	A list of all lineage strings
Devstage1.txt	Graph of the first development stage
Devstage2.txt	Graph of the second development stage
Devstage3.txt	Graph of the third development stage
Devstage4.txt	Graph of the fourth development stage
Devstage5.txt	Graph of the fifth development stage
Devstage6.txt	Graph of the final development stage
Nrsynapses.txt	Graph of the entire nervous system
Nrsynapsesgenes.txt	Graph of the entire nervous system with genes
RightNeurons.txt	Neurons on the right side of the nervous system
RightLineage.txt	Lineage strings on the right side of the nervous system

Folder: OutputFiles:

Bitgenes-output.txt	Sample output of bitgenes.cpp
Develstages-output.txt	Sample output from develstages.cpp
LeftStrongComp.txt	Strong components of the left side of the nervous system
LeftSynapses.txt	Synapses on the left side of the nervous system
LinDistCL-output.txt	Sample output of LinDistCl.c (matrix file)
Linsyn.xls	An Excel file of lineage and synaptic distances
Neuronenvgenes-output.txt	Sample output of neuronenvgenes.cpp
Neuronenv-output.txt	Sample output of neuronenv.cpp
NeuronDescriptions.txt	Descriptions of each neuron
Neuronpaths-output.txt	Sample output of neuronpaths.cpp
Pathgenes-output.txt	Sample output of pathgenes.cpp
Pathgenespairs-output.txt	Sample output of pathgenespairs.cpp
RightStrongComp.txt	Output of strongcomponents.cpp for right side
RightSynapses.txt	Synapses on the right side of the nervous system
StrongComponents-output	Output of StrongComponentsCL.cpp
Subpath-analysis.xls	Spreadsheet analysis of subpaths output
Subpaths-output.txt	Sample output of subpaths
SynDistCL-output.txt	Sample output of SynDistCL.cpp (matrix file)

Appendix VII
Bibliography

1. "Path Matching and Graph Matching in Biological Networks," by Qingwu Yang and Sing-hoi Sze. Journal of Computational Biology, Volume 14, Number 1, 2007.
2. "Dynamic Plasticity Influences the Emergence of Function in a Simple Cortical Array," by Jeff Shrager and Mark H. Johnson, Neural Networks, Volume 9, Number 7, pp. 1119-1129, 1996.
3. "Cortical Localization of the G-alpha Protein GPA-16 Requires RIC-8 Function During *C. elegans* Asymmetric Cell Division," Afshar et al., Development, 132, 4449-4459, 2005.
4. "The Embryonic Cell Lineage of the Nematode Caenorhabditis elegans," J.E. Sulston et al., 100, pp. 64-119, 1983.
5. "Asymmetric cortical and nuclear localizations of WRM-1/Beta-Catenin during asymmetric cell division in *C. elegans*," Takeshita et al., Center for Developmental Biology, Division of Bioinformation, Kobe University, Kobe, 650-0017, Japan.
6. ***"Computational Inference of the Molecular Logic for Synaptic Connectivity in *C. elegans*," Bioinformatics, Volume 22, Issue 14, ISMB (Intelligent Systems in Molecular Biology) 2006, pp. e497-506.
7. ***http://www.ee.columbia.edu/~anastas/ismb2006/ is the website that is the source of the three files describing the synaptic network which is the nervous system of *C. elegans*: Post_Synaptic_Neuronal_Partners.xls, Pre_Synaptic_Neuronal_Partners.xls, and Gap_Junction_Partners.xls.
8. "Wiring optimization can relate neuronal structure and function," by Chen, Hall, and Chklovskii, PNAS, March 21, 2006, Volume 103:12 pp. 4723-4728 (This paper is the source of the adjacency matrices described in 7. above).
9. http://www.wormatlas.org/neuroimageinterface.htm is the source of the neuronal lineage string information. This information is also available through the Pedigree browser of WormBase.
10. "The Structure of the Nervous System of the Nematode Caenorhabditis Elegans," J.G. White, et al., Philosophical Transactions, Royal Society, London, Series B, Biological Science, Volume 314, Issue 1165, November 12, 1986, pp. 1-340. This paper has an associated website, 'The Mind of a Worm:' http://www.wormatlas.org/MoW_built0.92/toc.html.
11. Data Structures with C++ Using STL. Second Edition. William Ford and William Topp. Copyright 2002, Prentice-Hall.
12. C++ Programmer's Guide to the Standard Template Library. Mark Nelson. IDG Books. Copyright 1995.
13. "The Posterior Nervous System of the Nematode Caenorhaditis: Serial Reconstruction of Identified Neurons and Complete Pattern of Synaptic Interactions," Hall and Russell, Journal of Neuroscience, 1991.

14. "Neuroanatomy: A second look with GFP reporters and some comments," Hobert and Hall, October 1999, WormBreeder's Gazette
15. AY's Neuroanatomy of *C. elegans* for Computation (CRC Press, Boca Raton). Durbin et (1986), and Achacoso, T. B., and Yamamoto, W. S. (1992).
16. "Post-embryonic Cell Lineages of the Nematode Caenorhabditis elegans," Sulston and Horowitz, Developmental Biology, 1977 (Volume 56: 110-156).
17. "Post-embryonic Cell Lineages of the Nematode Caenorhabditis elegans," Sulston and White, Developmental Biology, 1983 (Volume 100: 64-119).
18. "Network flow and testing graph connectivity," Even et al., SIAM J. Computing, December 1975,
19. "Flows in Networks," Ford et al., Princeton University Press. 1972.
20. Discrete Computational Structures, Korfhage, Academic Press, New York, 1974.
21. "Maximum Flow in Planar Networks," Itai et al., SIAM J. Computing, 1979.
22. "Efficient determination of the transitive closure of a directed graph," Munro, Information Processing Letters, 1971-1972.
23. Graphs and Their Uses, Ore, Random House, 1963.
24. Graph Theory, 1736-1936. Biggs, Lloyd, and Wilson. Oxford University Press.
25. "Computational inference of the molecular logic for synaptic connectivity in *C. elegans*," Bioinformatics, Volume 22, November 14, 2006, pages 3497 to e506, done at the Center for Computational Biology and Bioinformatics at Columbia University.
26. Chklovskii et al. "Search for computational modules in the *C. elegans* brain," BMC Biology, 2004, 2:25.
27. Milo et al. "Network motifs: simple building blocks of complex networks" (Science, 2002).
28. "Computational modeling of *C. elegans* vulval induction", Sun and Hong, Bioinformatics, Volume 23, ISMB/ECCB 2007, pp. 499-507.
29. "Computational insights into *C. elegans* vulval development," Fisher et al., PNAS, February 8, 2005, Volume 102),
30. "Computer simulation of the cellular arrangement using physical model in early cleavage of the nematode *C. elegans*", Kajita et al., Bioinformatics, volume 19, number 6, 2003, pp. 704-716),
31. "Application of machine learning and visualization of heterogeneous data sets to uncover relationships between translation and developmental stage expression of *C. elegans* mRNAs", Trutschl et al., Physiological Genomics, Volume 21, February 8, 2005, pp 264-273.
32. "Non-EST-based prediction of novel alternatively spliced cassette exons with cell signaling function in *C. elegans* and human," LeParc et al., Nucleic Acids Research, 2007, Volume 35, Number 10.
33. "Identification of a novel cis-regulatory element involved in the heat shock response in *C. elegans* using microarray gene expression and computational methods," Thakurta et al., Genome Research, 2002, Volume 12, pp. 701-712.
34. "Glia Promote Local Synaptogenesis Through UNC-6 (Netrin) Signaling in C. elegans**."** Daniel A. Colón-Ramos, Milica A. Margeta, and Kang Shen,

Science**,** Oct 2007; 318: 103 - 106.
35. “Spatial Regulation of an E3 Ubiquitin Ligase Directs Selective Synapse Elimination**,”** Mei Ding, Dan Chao, George Wang, and Kang Shen, Science**,** Aug 2007; 317: 947 - 951.