COMP 5206 Final Project

Matthew MacLeod
100771589

Overview

The primary goal for the project was to implement a system like that described by Jenkins and Stekel (2010) for evolving gene regulatory networks. This study presents enough detail to make the task of replicating it fairly straightforward. Once the basic functionality of the system was in place, I began working on adding facilities for data collection and analysis. A number of measures are gathered throughout a run and code is provided to plot and review the data afterwards. Overall, the results are similar to those presented in the original study. My motivation for the project was primarily to explore the workings of the system and gain some experience with the methodology of *in silico* evolution, as this is likely to figure prominently in my thesis work.

For convenience, the original paper (Jenkins & Stekel, 2010) is included with the submitted package. Since the implementation presented here remains as close as possible to the original, and since they describe their work much better than I could, I will not waste time describing the system here. (The Materials and Methods section provides a close look at the model components, simulation procedure, and evolutionary framework, and I have made every reasonable effort to follow this description exactly.) Thus, the implementation here has thirteen input genes (*fod*1-9, *nrg*1-2, *rcp*1-2), six output genes (*syn*1-4, *rsp*1-2), and a number of regulatory genes. While the original study was interested in comparing stressed vs unstressed environments, I simply implemented the stressed version of the environment because I was most interested in exploring the more biologically realistic system.

The main results presented by Jenkins and Stekel show that: i) network structure and function are directly related to environmental conditions; ii) network complexity arises in stages, generating a hierarchy of connected sub-systems; and iii) 'global regulators', or transcription factors that regulate a large portion of the genome, arise through adaptive selection (as opposed to neutral evolution). While my main concern was not with addressing any particular question or prediction, the results clearly show

the same kinds of trends as found in the the original, although I have not performed any quantitative analysis.

Once the basic system was up and running, it became apparent that food availability is a central determinant of the evolution and the structure of the resulting networks. Unfortunately, this is also the only aspect of the system that is not well-described by Jenkins and Stekel (at least, not in this paper or any of the others I have read). The stressed environmental condition in the original calls for each of the nine food sources to be randomly available for approximately 12% of the simulation. My implementation simply generates a random number from [0, 1] for each food type at each timestep and makes the food available if the random number is less than the corresponding parameter. With each food set to 0.12, there is too much energy available to the model so that it is fairly easy for models to survive despite having many genes and inefficient regulatory systems. After some experimentation, I found that a value of 0.08 produced results like those in the original.

To further examine this effect, I compared networks generated with all foods available with probability 0.12 and 0.08. I also ran some experiments in which there is a base probability for all foods, but four of them are randomly selected to be available at twice the base probability and the other five at one half of the base. The foods were randomly selected every 100 generations, providing a regular cycle of relatively major disruptions to the food supply. This regime of 'flipping' the probabilities every 100 generations was applied with a base probability of 0.12 and 0.08, which I arbitrarily called 'easy' and 'medium'. Thus, there are four test conditions: two with constant food probabilities (easy, 0.12; medium, 0.08), and two with variable or 'flipping' probabilities (again, easy and medium with base probabilities of 0.12 and 0.08, respectively). The results are discussed below, but they generally support the findings of Jenkins and Stekel in a different way, by suggesting that networks progressively evolve to be better able to withstand these shifts in food availability.

Instructions

The code is written in Java using Eclipse. The main method is in the file called *Main.java* in the root package (called '*jas*' for *J*enkins *A*nd *S*tekel) and simply displays the user interface. Aside from a few very basic presets, the user interface has little control over system settings. In this regard, the most important file is *Params.java* (in the *jas.sys* package) which, as its name suggests, stores the parameters for pretty much everything. All of the parameters from the original paper are here, along with some others that were required to make the system work and for data collection. This file is organized into three sections (model, simulation, and GA parameters) and has many explanatory comments. The main parameters are identical to those from the paper and I have tried to keep to a similar naming scheme for those I added. The only entry that really needs to be adjusted is *maxGen,* the maximum number of generations for evolution. As a general rule, a few hundred generations will take a few minutes, but as models start surviving longer, the simulation time increases dramatically. A full 10,000 generation run takes at least a couple of hours (a fair bit longer than the 90 minutes reported by Jenkins and Stekel). Fortunately, 1000 or 1500 generations will run in 15 or 20 minutes and usually produces some interesting results. The value is currently set at 1000. Currently, it cannot be set from the GUI.

Other parameters you may wish to adjust are those relating to data collection intervals, of which there are two. Throughout a run, the system stores the best model every *iSampleModel* generations and stores the gene regulatory network of the best model every *iSampleGRN* generations. These parameters are found near the end of the Params.java file and are currently set to 50 and 100, respectively, which seems to be sufficient.

Upon running Main.java, the GUI is displayed. There is a menu item at the top left for very basic *presets*. These set food availability parameters to those like the test conditions described above but do not affect any other parameter values (such as maxGen). You can get a good sense of what each of these presets will do with a maxGen value of 1000 – 2000. At the moment, the presets are the best

way to set food availability parameters. Selecting a food availability preset from the menu will create and initialize the system. To start the run, simply click the 'Go' button. (The other buttons there are pretty intuitive). There is a check box that specifies if the system will collect data or not (default is checked). Throughout the run, the best and worst model in the population have their pertinent data displayed on the left side of the screen, and the gene regulatory network of the best model is displayed in the centre (roughly). The input genes are labelled and shown in green at the top, regulatory genes in grey in the middle, and output genes labelled and in red at the bottom. Activating edges are drawn in red and repressing edges in blue. Additionally, the brightness and thickness of the red or blue line indicates the strength of the binding. The direction of the interaction is represented (rather clumsily) by starting the line just outside the node that initiates the interaction (i.e., by its protein binding to the DNA of the target); at the target end, the line continues into the body of the node. (For example, if the input gene *rcp1* activates the output gene *rsp1*, there will be a red line beginning just below the *rcp1* node and ending inside the *rsp1* node.)

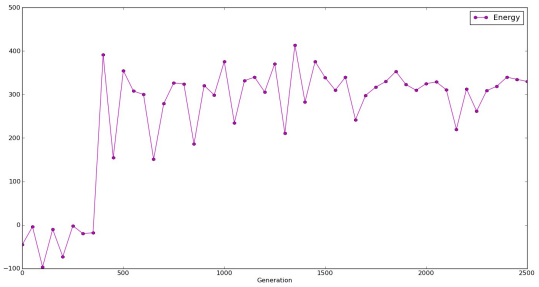
At the end of a run (if the collect data box has not been unchecked), a new window will appear entitled “GRN Viewer”. As its name suggests, this window displays the gene regulatory networks of the best models, collected every Params.iSampleGRN generations. The GRNs are contained in the list on the left hand side; selecting one will draw it in the centre panel. There is a 'Save image' button that allow you to save a .png image of the currently selected GRN. Additionally, there is a directory called 'data' in the main 'Code' directory, and the system will output three files to this directory (when collect data is checked): modelStats.txt, shapeStats.txt, and shapeDistros.txt. All of these files contain data collected from the best models every Params.iSampleModel generations. The first two are used by the Python script to plot various measures from the models (e.g., fitness, biomass, energy, number of genes, etc.) and also data relating to the 1D shape space (e.g., most common shape, ratio of populated to unpopulated shapes, mean shape, etc.). The third file (shapeDistros.txt) is just a printout of the frequency distributions of the 128 shape values for each of the collected models. It is included because

I was interested in how these distributions evolve over time, but I am unsure of what to do with them, so they are just there to look at if you are interested. The Python script (plotData.py, also in the 'data' directory) requires matplotlib to run, and will parse the data from the files and display a number of figures. (Note that the system will overwrite any data files it finds in the 'data' directory, so one should move any files one wishes to keep before starting a new run).

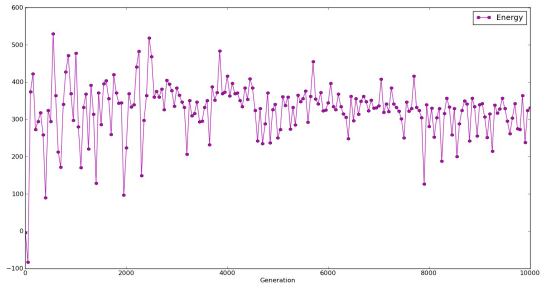
Results

This section presents a sample of the figures produced by the plotData.py script and a brief discussion of each, for each of the four test conditions (labelled Easy, Medium, Easy Flip, and Medium Flip, as indicated above). (Full size images can be found in the *Code/data/Examples* directory.)

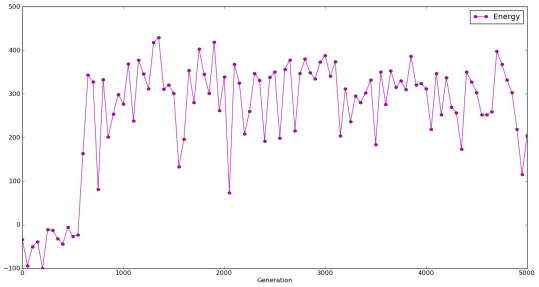
Energy



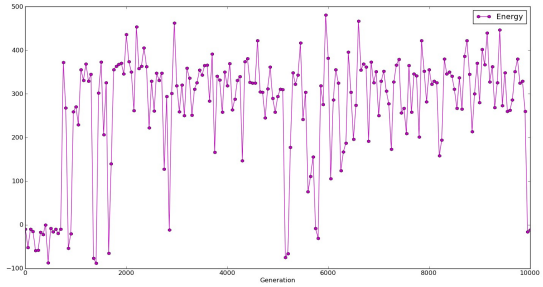
Easy



Easy Flip



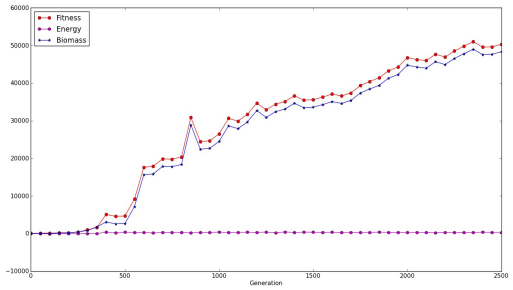
Medium



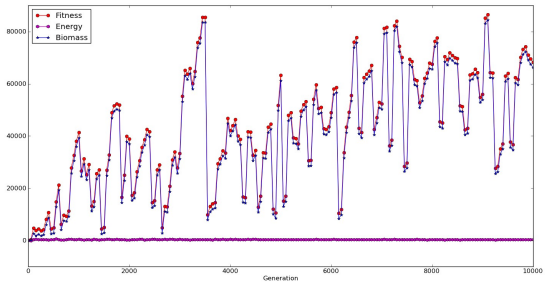
Medium Flip

There is not much to note here, except that there is generally an initial period during which models cannot manage energy and frequently die from lack of energy (starvation). Then there is a sudden jump as an effective energy regulation system arises. It also seems as though there is a slight general trend toward fewer and less extreme fluctuations in the later generations.

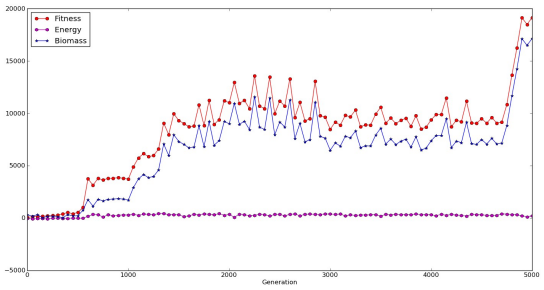
Fitness, Energy, and Biomass



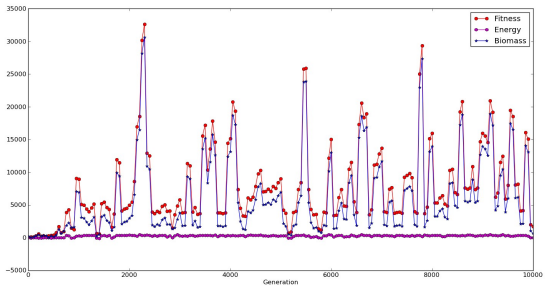
Easy



Easy Flip



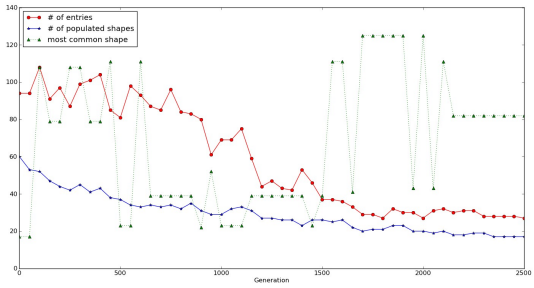
Medium



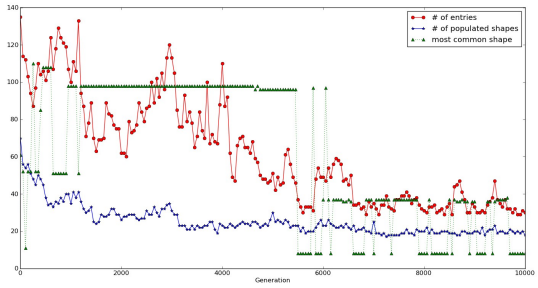
Medium Flip

These are among the most informative graphs. They clearly show why 0.12 probability for all foods is 'Easy'. The Medium condition shows clear evidence of the stages of network evolution, with stress response preceding energy regulation preceding growth systems preceding global regulation. In the Flip conditions, there seems to be a trend toward greater robustness against the shifts in food availability. This is more prominent in the Easy Flip condition, but it is apparent in the Medium Flip as well.

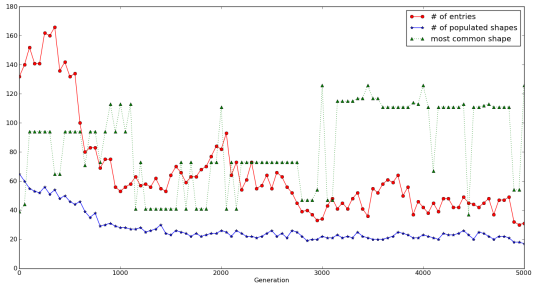
Shapes



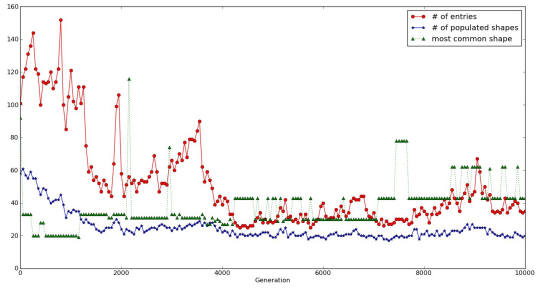
Easy



Easy Flip



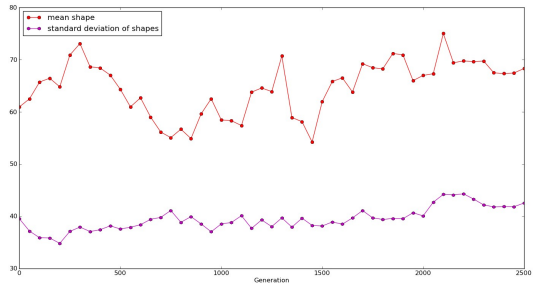
Medium



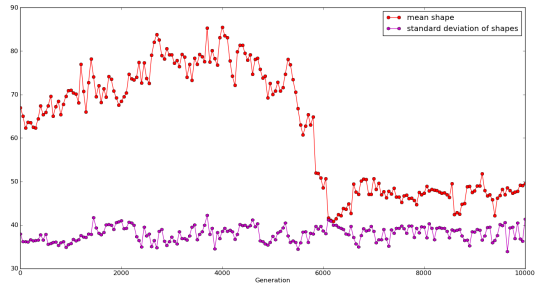
Medium Flip

These graphs plot data concerning the 1D shape space. (Red is the number of shaped items, blue is the number of populated shapes, and green is the most common shape). As expected, there is a general trend towards fewer genes and binding sites, but the fluctuations in each have distinct characteristics. I am not sure what that means. Also as expected, the number of populated shapes decreases over time to some fairly stable minimum. This trend seems to hold across the conditions. I am not sure if the most common shape says anything interesting.

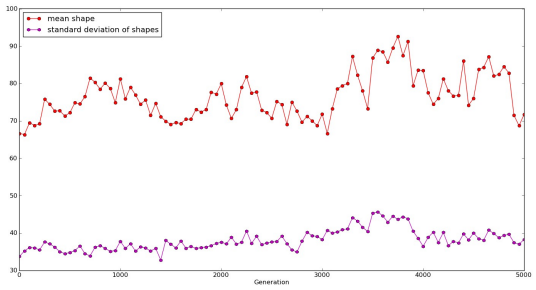
Mean and standard deviation of shapes



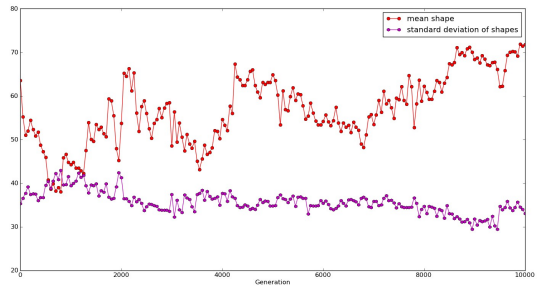
Easy



Easy Flip



Medium



Medium Flip

These graphs show the 'mean' shape (calculated over the populated shapes and weighted by the frequency of each shape) in red, and the standard deviation of shapes in purple. Here again, I am not sure what, if anything, these results mean, but it is interesting that both Flip conditions have overlapping regions while the non-Flip conditions have neatly separated plots. There wasn't time to compare a number of runs under each condition.