Ingeneue Tutorial: The Segment Polarity Model

This tutorial is intended to get you started with our segment polarity network model. We used this network as our test case to develop Ingeneue. Before doing this tutorial, you will want to look at the paper we published on this model, as the tutorial essentially tells you how to recreate many of our results in that paper:

G. von Dassow, E. Meir, E. Munro, G. Odell. (2000). The segment polarity network is a robust developmental module. *Nature* **406:** 188-92

The tutorial takes you through the following steps:

- loading a network file and running the model defined in that file
- loading a file with pre-found sets of parameters under which the network will produce a desired behavior
- sampling random parameter sets
- changing parameter values by hand
- changing initial conditions

For many of these steps there are more complicated and flexible ways of doing the same thing which you may want to read about in the full manual after finishing the tutorial. The next tutorial after this one shows you how to construct your own network files.

Disclaimer: While we expect to make Ingeneue a polished program with a nice interface in the future, up until now it was just designed to be used by its developers. Ingeneue is thus a finicky program, and we have just gotten good at avoiding its quirks. We frankly have been more concerned with getting the core algorithms right than with making a spiffy user interface. Moreover, Java is not yet as mature as it could be. So all in all, please be forgiving and expect to quit and restart the program several times. We are happy to hear about bugs and other frustrations and we will try to fix problems you tell us about.

Starting up and running a model

1) Look at the ReadMe file that came when you downloaded the program and install the program on your machine as specified in that file. In particular, remember that you need to have the swing libraries installed, as explained in the ReadMe file.

2) Run Ingeneue by:

   - (Macintosh): double-clicking on the program icon.
   - (Unix): assuming an appropriately-name Java executable is installed and in your path, type "java main.GeneNet &" – depending on the version of Java you use, you may need to include a flag to invoke the just-in-time compiler.
   - (Windows): we don't use Windows so you'll have to figure this out yourself, but it should be more or less the same as Unix.

   Two windows should open; the lower one is a console window where the program prints out status messages (see Figure 1). The other is the main control window, including the main menus. Those of you used to Macintosh computers may be momentarily confused to see the menus are inside the window, not in the menu bar. This is because

Java programs are intended to run identically on all platforms. *Note: if one or the other windows fails to appear, kill the program and restart. For some reason the most recent version of Apple's MRJ occasionally fails to display the windows on startup.*

Until we develop a more sophisticated interface, Ingeneue uses text files as inputs. This tutorial should have come along with a folder called Inputs, which contains files of three types: ".net" files define genetic networks; ".params" files contain sets of parameters for a particular network; ".iter" files contain instructions for automatically running models within Ingeneue. Before loading a network, take a quick look at one of the network files.

3) Using any text editor, open the file "spg1_basic.net" ("spg" = segment polarity genes). Skim through this file to see the basic structure.

We'll come back to parts of this file later on in more detail. For now, just notice that the top section defines all the different components of the model – mRNA's, proteins, protein complexes and so on. These components are the "Nodes" of the network. Next comes a section specifying how each Node changes over time. The items in this section correspond to equations that will be summed into one overall differential equation per Node. We call these equation fragments "Affectors". The last two major sections give the values for each parameter in the model and the initial conditions.

4) Close the "spg1_basic.net" file.

5) Select 'Load' from the 'File' menu in Ingeneue, and load the "spg1_basic.net" file you were just looking at.

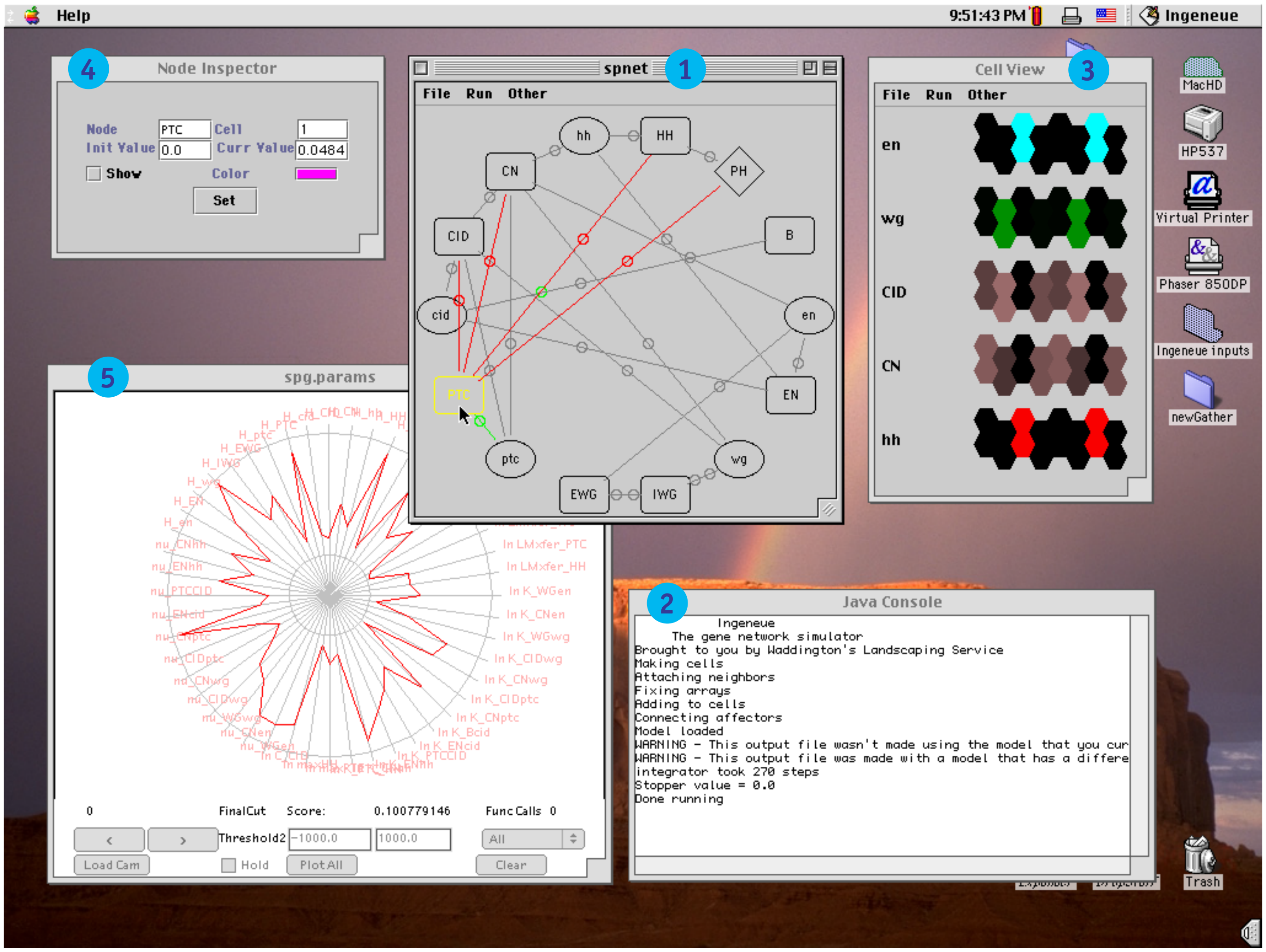
In addition to the two windows that appeared at start-up, Ingeneue opens a new window titled "Cell View". Cell View contains a column of pictures of the field of cells that will be used in this model (rightmost window in Figure 1). Each picture is labeled with one of the Nodes in the model, and will show the concentration of that Node in each cell. Cell View only shows a subset of the Nodes, and you can choose which ones in the input file or while the program is running (see below). To make things run fast the model is just a two-segment strip of cells and only two cells high. The initial conditions don't show up in the Cell View until you reset or run the model. Start by resetting so you can see what the initial conditions look like.

Also, the main control window now contains a diagram of the network, showing which Nodes influence each other directly. Clicking on each Node brings up a Node Inspector window that allows you to specify whether that Node is shown in the Cell View or not, and, when the model is running, the Node Inspector allows you to see the numerical value of a Node in an individual cell.

6) To see the initial conditions, reset the model by selecting 'Reset' from the 'Run' menu.

7) Run the model by selecting 'Run' from the 'Run' menu.

Pretty boring, huh? Obviously the parameter values specified in the input file aren't too conducive to stripe patterning. The next section shows you how to load a file with sets of parameters (that we pre-selected) which cause the model to make stripes.



Node: PTC Cell: 1
Init Value: 0.0 Curr Value: 0.0484
Show Color
Set

0 FinalCut Score: 0.100779146 FuncCalls 0
< > Threshold2 -1000.0 1000.0 All
Load Cam Hold Plot All Clear

```
        Ingeneue
    The gene network simulator
Brought to you by Waddington's Landscaping Service
Making cells
Attaching neighbors
Fixing arrays
Adding to cells
Connecting affectors
Model loaded
WARNING - This output file wasn't made using the model that you cur
WARNING - This output file was made with a model that has a differe
integrator took 270 steps
Stopper value = 0.0
Done running
```

Figure 1. The Ingeneue windows. The main control window (1) is in the middle, and displays a diagram of the currently loaded network; the window title is the name of the network specified in the model file. The selected Node is highlighted in yellow; green lines show which other Nodes affect the selected one, whereas red lines show which other Nodes are affected by the selected Node. Circles in the middle of the connecting lines allow access to the parameters governing the Affector representing that link. The Cell View (3) displays concentrations of gene products (Nodes) in the entire cell grid. The Inspector window (4, currently displaying information about the selected Node), to the left, shows the numerical value of an individual Node and allows one to set initial concentrations and whether the Node is shown in the Cell View. Ingeneue uses the Java Console (2) to issue error messages or to announce other information. Finally, the Parameters window (5), which titles itself according to the parameter file loaded, in this case "spg.params", has controls for inspecting and applying parameter sets to the model.

Using pre-collected parameter sets

8) Select 'Load' from the 'File' menu and load the file "spg.params".

The program will probably show you some warning messages in the console window. Ignore these unless they seem really dire; it should be telling you that these parameter sets were found with a different input file than you presently have loaded, which in this case is just fine. Ingeneue will then bring up a new Parameters window that contains a plot that looks a little like an incompetent spider's web (bottom right window in Figure 1). This plot shows you one or more parameter sets. *Note: you will* *need to expand the window to see all the controls and to see the wheel plot clearly.* Each spoke of the wheel is the axis for a single parameter, with the lowest possible value that parameter could take on at the inner circle and the highest value at the outer circle. A parameter's value is plotted as a point on its spoke in between the inner and outer circles. There are 48 parameters in this model so there are 48 spokes. A single parameter set is then a polygon with a vertex on each spoke at the position of each parameter's value in that set. Don't worry about reading the graph precisely; the point is that the different polygons are vastly different from one another in shape, and thus parameter sets are widely distributed in parameter space.

9) To get a feeling for how the parameter sets vary, scan through the parameter sets by using the arrow buttons at the bottom of the Parameters window. Again, the point to recognize is that, although as you will soon discover all these parameter sets confer similar behavior on the model, they specify widely different values for each parameter.

10) Pick one of the parameter sets and load it into the model by clicking on the 'Load Cam' button in the Parameters window.

11) Choose 'Run' from the 'Run' menu in either the Cell View or main control windows, to run the model with this new set of parameters.

Much better, yes?

Note: Figure 1 shows what things should look like at this point; the pattern in the Cell View window was made with the first parameter set in the "spg.params" file.

12) Repeat steps 10 and 11 for as many of the other parameter sets as you can. Notice how all these parameter sets make *qualitatively* similar patterns with respect to engrailed (en), wingless (wg), and hedgehog (hh). At the same time, notice how different the *detailed* behavior of each parameter set is, especially during the initial period, despite all making qualitatively the same pattern.

If you want to sit back and relax and watch the program test all the parameter sets in turn, load the file "tester.iter" and choose 'Run Iterator' from the 'Run' menu.

So far you have seen a single instance of a parameter set that does not form segments and several instances of parameter sets that do form segments. What else can the model do? The next section explores that question by running the model with randomly generated parameter sets.

Sampling random parameter sets

To run a model automatically in Ingeneue you use a software module called an Iterator. An Iterator performs some predefined set of actions over and over again. One of the simplest Iterators picks random sets of parameters for a model and runs it, usually using a pattern recognition module to score the behavior of the model with each parameter set. The framework we have for Iterators right now is a bit more complex than it ought to be, and this tutorial won't try to explain them, just have you use a simple one.

13) Select 'Load' from the 'File' menu and choose the file "randomsampler.iter".

14) Ingeneue may or may not like loading an iterator file after you've already been playing around with parameters so long. If you see lots of messages about exceptions in the

Console window, just quit the program, restart it, reload the "spg1_basic.net" file, and then load "randomsampler.iter".

15) Select 'Run Iterator' from the 'Run' menu.

Ingeneue cycles through one randomly-picked parameter set after another, running each 200 minutes. Here are several things to pay attention to:

- Some runs are slower than others, but all cover roughly 200 minutes (time is shown in the upper left of the Cell View). The speed difference reflects differences in the "stiffness" of the equations, which depends on the particular parameter values in the current set. This speed difference in the simulation has absolutely nothing to do with the speed at which the pattern could be formed in a real embryo, just with how fast the computer can solve the equations.
- Lots of runs do boring things like turn all genes off or turn something on everywhere. Many others look promising but then collapse. See if you can get a feel for what tends to happen as the initial pattern collapses. Look at the network diagram to see if you can intuit what's going on.
- There are many other stable patterns besides the segment polarity pattern that the network can adopt, even though they all start from the same initial conditions. You should see some of them going by, and every once in a while you may see the model make the correct pattern.

Note: for the segment polarity model we impose repeating boundary conditions, meaning that on each edge the cell grid wraps around so that, for instance, the cells on the east edge are actually neighbors of the cells on the west edge. The file "spg1_basic.net" specifies an 8x2 cell grid, which seems to be about right for a human to see clearly the nature of the patterns the model makes. However, as long as we impose repeating boundary conditions, and as long as the model is entirely deterministic (i.e. no random noise afflicts the Nodes), a 4x1 cell grid is equivalent but four times faster, which of course makes searches much more bearable for the human. For example, once you have a feel for the variety of patterns the model makes, you might want to get a feel for how frequent some of those patterns are. The file "spg1_4cell.net" uses a 4x1 grid, but is otherwise exactly the same as "spg1_basic.net". Load "spg1_4cell.net", then "randomsampler.iter" (you may have to restart the program if it emits a stream of error messages), and then choose 'Run Iterator' again from the 'Run' menu.

16) When you are tired of watching this, select 'Stop' from the 'Run' menu. You may need to quit, restart, and reload the model file again at some point below.

Until now you have been using either working parameter sets that we found, or random sets. The next section shows you how to set parameter values yourself to get a better feel for how each individual parameter might affect the behavior of the model.

Changing parameter values by hand

There are two different ways to change parameter values of an Ingeneue model. You can change them in the input file. You can also change them through the Ingeneue interface by clicking on the little circles in the network diagram displayed in the main control window, which shows all the parameters governing that link in the Inspector window. This is the incipient beginnings of our eventual full-blown point-and-click user interface to Ingeneue. You're welcome to click away on the network diagram (the last section will ask you to do this for changing initial conditions) and figure out how to change parameters that way (its about as straightforward as can be; see Figure 2). However, we've written this exercise as if you're still using the old-fashioned method of editing the text file, which has the advantage that you can go in and quickly edit lots of values without all the clicking around to find the one you want.

17) Start with a parameter set that makes the right pattern by loading the network file "spg1_01.net".

18) Run the model to see what pattern these parameters produce, taking note of the initial transients, and how well it matches the target pattern.

19) Open "spg1_01.net" in a text editor and scroll to a section labeled "&ParameterValues".

    You'll see a bunch of lines that look like this:

```
&K_WGen   0.1687772 0.0010    1.0  Logarithmic
```

    The tag with the ampersand is the name of the parameter. The first number is the set value of that parameter. This is the only number we'll work with in this tutorial. The next two are the lower and upper bounds for that parameter. The "Logarithmic" specifier means that this parameter is plotted on a log scale in the Parameters graph, and that during random sampling it varies on a log scale.

20) To figure out what each of the model's 48 parameters does, peruse "SPGParameters.pdf" (which should have come with this file). You will find brief descriptions of each of the parameters in that document. The manual and other documents contain fuller descriptions of the general classes of parameters used in Ingeneue models.

21) Think about what parameter you might want to change to achieve a particular effect on the behavior of the model (or just pick randomly). Then replace the current value of the parameter – the first number on the parameter definition line – with another value between the minimum and maximum values. The minimum and maximum are merely guidelines, and may be violated if you wish. For example, for parameters starting with "K_", a value well above 1.0 (i.e. 10 or so) usually means that the regulatory connection governed by that parameter is too weak to do anything.

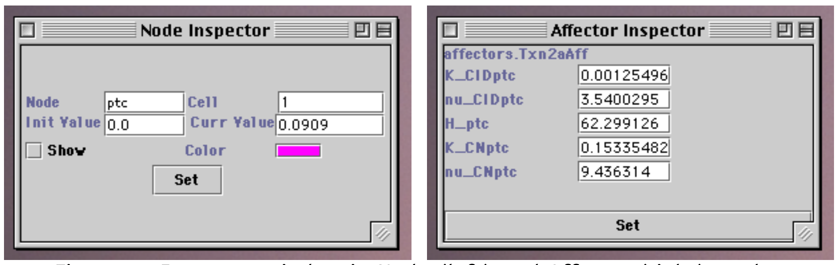
22) Choose 'Save As' in your text editor and save the file under a new name, to avoid overwriting the original.

23) Load this new file into Ingeneue and run it.

24) Repeat 21 - 23 a bunch of times until you are sick of the little colored hexagons. You can also start with another initial parameter set by using one of the other "spg1_XX.net" files. Depending on the parameter set you start with you may find it very hard to change the behavior of the model, whereas with another starting set you may find it very easy to nudge it into doing something quite different.

Changing initial conditions

The "initial conditions" for the model is the pre-pattern that the model starts running from – the initial concentrations of every Node in every Cell. It is the pattern displayed at time 0 in your runs above (or after resetting the model). As with changing parameter values, there are two ways to change the initial conditions in an Ingeneue model. You can change them in the network file, or you can change them through the interface. While changing through the network file is more powerful, in this case it is easier to change them through the interface. In this section, you explore the effect of simple changes to initial conditions.



Node: ptc
Cell: 1
Init Value: 0.0
Curr Value: 0.0909

affectors.Txn2aAff
K_CIDptc: 0.00125496
nu_CIDptc: 3.5400295
H_ptc: 62.299126
K_CNptc: 0.15335482
nu_CNptc: 9.436314

Figure 2. Inspector window in Node (left) and Affector (right) modes.

25) Load a network file (one of "spg1_XX.net") that makes nice stripes.

26) In the Cell View window, double-click on one of the cells for a Node whose initial value you want to change. For instance, if you want to change the initial concentration of engrailed mRNA in the top-left cell, double-click on the top left cell in the row labeled "en".

    Note: to correspond with the numbering of arrays in Java and similar programming languages, the first (that is, top left) Cell in the cell grid displayed in Cell View is numbered 0.

27) In the Inspector window which appears, there is a number called Init Value. Concentrations for most Nodes are scaled between 0 and 1, so change the Init Value to any number you want between 0 and 1.

28) Click on the 'Set' button in the Inspector window.

29) Reset the model. You should see the pre-pattern reflecting the change you just made.

30) Run the model.

31) If you want to change the initial value for a Node that is not shown in the Cell View, click on the Node in the network diagram and then click once (only once) on the cell where you want to change its value in any of the rows in the Cell View. You will now get the appropriate Inspector and can change the initial value as above. Or, click on the Node in the network diagram, then check the box to show the Node in the Cell View, and then double-click the appropriate cell in the new row.

32) Play around with different initial conditions. Start with one of the "spg1_XX.net" files, and then modify the initial conditions to explore what effect they have. How far can you get from the crisp initial condition that we used and still get good segments? What other patterns can you get with different initial conditions? You might want to modify the initial conditions in "spg1_4cell.net", then use the "randomsampler.iter" to conduct a random search.

A test of sensitivity to random noise

Next you'll test how sensitive the model is to random variations in the initial concentrations of engrailed and wingless. Of course, the model needs parameters, so we use the ones collected in "spg.params"; thus you'll really be testing the sensitivity of the model *with each of those parameter sets*.

33) Load the file "spg1_sloppystripe.net". Choose 'Reset' from the 'Run' menu. You should see a 4x3 grid of cells with engrailed and wingless stripes, but with random fluctuations in the level of each of them in each cell.

34) Load the same file again, choose 'Reset', and note that it now exhibits a different random variation in the initial prepattern.

    The program is set up so that the initial prepattern is randomized slightly each time the model is loaded, not each reset. This means that you can test a battery of parameter sets against the same noisy variant.

35) To do that, load the file "spg.params". Flip through the parameter sets, press 'Load Cam', and run the model. Depending on the initial prepattern, you should see that some sets allow the model to "correct" the initial sloppiness, but others fail in various ways.

36) To get a new sloppy pattern, re-load "spg1_sloppystripe.net"; you should not need to reload the battery of parameter sets. Flip through them again, applying each one, and running the model. Repeat for several re-loads of "spg1_sloppystripe.net".

37) You can use the file "tester.iter" to run through all the parameter sets in "spg.params". Load "spg1_sloppystripe.net", then load "spg.params", then load "tester.iter", and choose 'Run Iterator' from the 'Run' menu. To get a new sloppy pattern, reload "spg1_sloppystripe.net", and re-run the Iterator. *Note: Ingeneue may not tolerate this sort of thing; if it doesn't, quit, restart, and follow the entire sequence.*

    The Iterator specified in "tester.iter" will run the model with each parameter set for 1000 min, and assign it a score based on whether it makes decent stripes of engrailed, wingless and hedgehog. If you are really diligent and repeat this step several times, you can follow the output in the console window to keep track of which parameter sets tolerate which insults. You will find that some parameter sets are consistently able to correct against random variation in the initial pattern, but that some are more fragile to this kind of perturbation.

    This exercise uses a bit of a cumbersome way to test this sort of thing, but it does it using very simple tools. Ingeneue includes some more powerful facilities for doing this sort of thing, but more important, the design of the program makes it easy to add iterator objects that conduct very sophisticated tests that involve running the simulation over and over again while varying parameters or initial conditions.

That's the end of this tutorial. The next tutorial takes you through building a very simple model with Ingeneue. After that, you can look in the manual for more details. If you are a programmer, there is also a fair bit of documentation contained within the source code, including descriptions of each of the Affectors (pieces of equations). And of course, you can always email one of us. Enjoy.