Bad Networking

Computational social science, public policy and programming.

Controlling NetLogo from Python

Background

One of NetLogo's key features is that it is self-contained. The code, GUI and documentation are all a single file, written and run within the NetLogo software itself. This has the advantage of helping with replication[1]: the entire model is easy to share, and guaranteed to run identically on any other copy of NetLogo without needing to mess with compilers and dependencies. However, the one thing NetLogo isn't well-suited for actually analyzing model outputs. Generally, the way to go has been to export the model results to CSV files, then loading them in R, Python or even Excel for analysis.

Recently, Jan C. Thiele released RNetLogo, which is just what it sounds like: an R package which can be used to call NetLogo from within R. You can read the Journal of Statistical Software paper, or a good tutorial by Joseph Rickert. I also happened to have recently seen a very clever integration of NetLogo and an external GIS tool at my summer internship. This led me to wonder on Twitter whether NetLogo's future was at least partially as an API for modeling, which could then be called and used from different languages and tools.

I decided to see how quickly and easily I could get NetLogo working with Python, which is generally my programming language of choice. It turned out not to be too difficult to hack together a basic connection, but it *is* hacky. More software engineering (probably quite a bit more) would need to get done to turn this into a robust solution.

Tools

Here's what I used to get this done:

- Python 2.7, along with the IPython Notebook and Matplotlib
- NetLogo 5
- Py4J, which provides a way of connecting Python to the JVM
- Eclipse for the actual Java coding; it makes life easier, but isn't strictly necessary

The actual connection consists of a Java program which is holds an instance of NetLogo in 'headless' (that is, GUI-free) mode. The program is running a Py4J gateway server for Python to interface with it.

Most of the code is adapted from the Py4J tutorial and NetLogo API documentation.

The full code is available on GitHub.

Java Bridge

The Java program here is meant to provide an interface to the core NetLogo commands. For now, it only implements three methods: opening a model, running a NetLogo command, and reporting a single numeric result. Be sure to include both **NetLogo.jar** and **py4j.0.8.1.jar**.

```
1  package Py2NetLogo;
2
3  import py4j.GatewayServer;
4  import org.nlogo.headless.HeadlessWorkspace;
5
6  public class NetLogoBridge {
7
8      HeadlessWorkspace ws;
9
10     public NetLogoBridge() {
11         ws = HeadlessWorkspace.newInstance();
12     }
13
14     /**
15      * Load a NetLogo model file into the headless workspace.
16      * @param path: Path to the .nlogo file to load.
17      */
18     public void openModel(String path) {
19         ws.open(path);
20     }
21
22     /**
23      * Send a command to the open NetLogo model.
24      * @param command: NetLogo command syntax.
25      */
26     public void command(String command) {
27         ws.command(command);
28     }
29
30     /**
31      * Get the value of a variable in the NetLogo model.
32      * @param command: The value to report.
33      * @return Floating point number
34      */
35     public Double report(String command) {
36         return (Double)ws.report(command);
37     }
38
39     /**
40      * Launch the Gateway Server.
41      */
42     public static void main(String[] args) {
43         GatewayServer gs = new GatewayServer(new NetLogoBridge());
44         gs.start();
45         System.out.println("Server running");
46     }
47 }
```

Now, when you run this program, it should start up a gateway server that can be called from within Python.

Python Interface

To connect to the bridge from a Python interpreter is even simpler. (This section was all executed interactively in an IPython notebook, which you can follow along in as well).

With the server running, execute:

```
from py4j.java_gateway import JavaGateway

gw = JavaGateway() # New gateway connection
bridge = gw.entry_point # The actual NetLogoBridge object
```

Sample Analysis

Following the example of the NetLogo API documentation and the RNetLogo tutorial, here's a a quick analysis on NetLogo's forest fire model from within Python.

First, we open the model itself:

```
# Path to Forest Fire model:
sample_models = "/Applications/NetLogo 5.0.2/models/Sample Models/"
forest_fire = "Earth Science/Fire.nlogo"
# Now, open the model file
bridge.openModel(sample_models + forest_fire)
```

Next, run the commands to set the forest density at 62%, give the model a fixed random seed for replication, set it up, and run for 50 steps:

```
bridge.command("set density 62")
bridge.command("random-seed 0")
bridge.command("setup")
bridge.command("repeat 50 [go]")
```

Now, to report the number of trees burned:

```
burned_trees = bridge.report("burned-trees")
print burned_trees
# If everything is working, this should print 4256.0
```

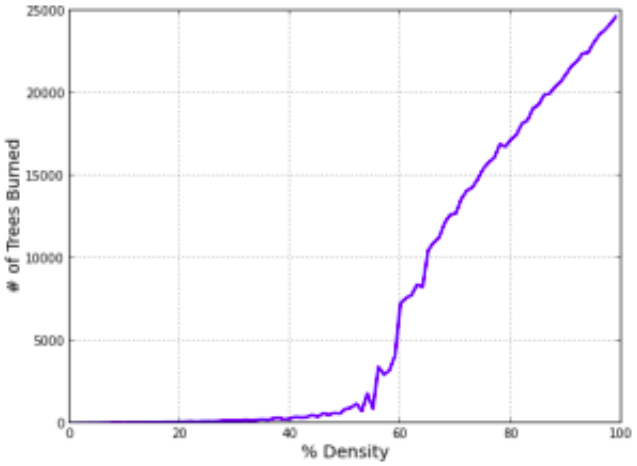
So far so good, but not very useful. A better thing to do would be to parameterize the entire thing, so that we could quickly vary the initial conditions and run the model from there.

```
def run_model(bridge, density, steps):
    '''
    Run the forest fire model, and return the number of trees burned.

    Args:
        bridge: The NetLogoBridge Java object
        density: Integer density percent, from 0 to 100
        steps: How many steps to run

    Returns:
        The number of trees burned, as a float.
    '''
```

```
    bridge.command("set density " + str(density))
    bridge.command("setup")
    bridge.command("repeat " + str(steps) + " [go]")
    return bridge.report("burned-trees")
```

Now, with this function, we can run the forest fire model Pythonically and plot the results with Matplotlib.

```
burned_trees = [run_model(bridge, i, 100) for i in range(0,100)]
fig, ax = plt.subplots(figsize=(8,6))
ax.grid(True)
ax.set_xlabel("% Density")
ax.set_ylabel("# of Trees Burned")
plt.plot(burned_trees, linewidth=2)
```



Conclusions

Python *can* be made to interface with NetLogo, albeit with a slightly inelegant Java intermediary. One way to avoid that is to use Jython to call the NetLogo API directly, but that would require giving up on most of the scientific Python stack. It's probably also possible to write a Python library which automatically launches a bridge server when it needs it, but that's outside-of-scope for me right now.

Nevertheless, I'm definitely going to keep this technique in mind for myself for next time I need to work with a NetLogo model. Besides just streamlining the data extraction process, I can imagine some powerful uses for a Python-NetLogo interface. For example, a function which calls NetLogo can itself be fed into one of Scipy's optimizers in order to identify what parameters maximize or minimize some output (see the IPython notebook for an example). We could also write our own genetic algorithm or other custom way to explore the parameter space. There could even be a fancy multi-level model, where some agents written in Python use a NetLogo model to drive their cognition. Even if nothing else, being able to run the model and analysis together within an IPython Notebook can help with record-keeping, documentation, transparency and replicability.

1. I'm not going to get into the debate over whether 'replication' in modeling should consist of running the same code on a different computer, or reimplementing the model based on the underlying theory and logic. ↩

This entry was posted in Uncategorized on July 26, 2014 [http://davidmasad.com/blog/netlogo-from-python/] .