V_Boole v2.11

Paul Meara

The Manual

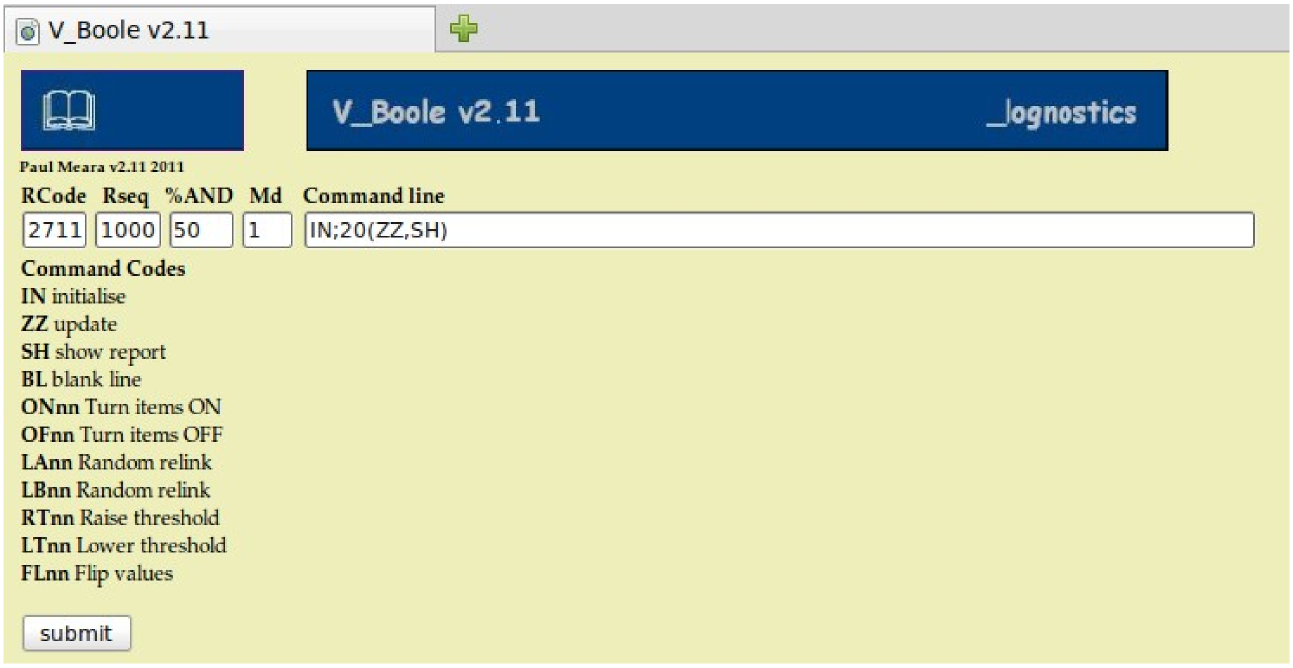
Introduction.

This manual is a short explanation of the commands that you need to operate V_Boole v2.11. For a fuller explanation of the ideas behind this program see: **PM Meara** *Boolean Lexicons*. You can download the current version of this paper from the **_lognostics virtual library**: http://www.lognostics.co.uk/vlibrary/.

Part One

1: V_Boole is a simple Boolean Network simulator. It lets you model produces some basic features of a lexical network, and explore how these features might relate to what we find in real vocabularies. V_B oole v2.11 works with a standard size vocabulary of 100 words and a standard set of commands. Its main purpose is to familiarise you with the way small Boolean networks function. Practice with V_Boole v2.11 before you move on to the more complicated models on this site.

2: The V_Boole v2.11 workspace looks like this:



Paul Meara v2.11 2011

RCode	Rseq	%AND	Md	Command line
2711	1000	50	1	IN;20(ZZ,SH)

Command Codes
IN initialise
ZZ update
SH show report
BL blank line
ONnn Turn items ON
OFnn Turn items OFF
LAnn Random relink
LBnn Random relink
RTnn Raise threshold
LTnn Lower threshold
FLnn Flip values

3: The simplest way in to V_Boole v2.11 is to type the following instruction into the data box labelled **Command line**

IN;SH;

The instruction is case sensitive, and the punctuation is important.
Your instruction consists of two parts:

IN; tells the program that you want to **Initialise** a new network. Technically speaking, the **IN** command sets up a random Boolean Network where N=100 and k=2. In ordinary language, this means that V_Boole v2.11 sets up a standardised network that you can experiment with. Since V_Boole v2.11 is a training tool, rather than a fully developed research program, it makes a number of simplifying assumptions at this point. It sets up a standard-size network of 100 nodes. Each of these nodes is randomly connected to two other nodes in the network. Thus, node 27 might be connected to node 56 and node 31, for example. Each node is randomly turned **ON** or **OFF**. Each node is also assigned a **type** which determines how it is turned ON or OFF by its inputs. V_Boole v2.11 has two types: AND items are difficult to turn ON. They go ON only if both their inputs are ON. OR items are more easily activated. They turn ON if one or more of their inputs are ON. All these properties are determined randomly.

SH; tells the program to show you the current state of the network.

4: Click on the button labelled **SUBMIT**. This will send your commands to the server. The output will look something like this.

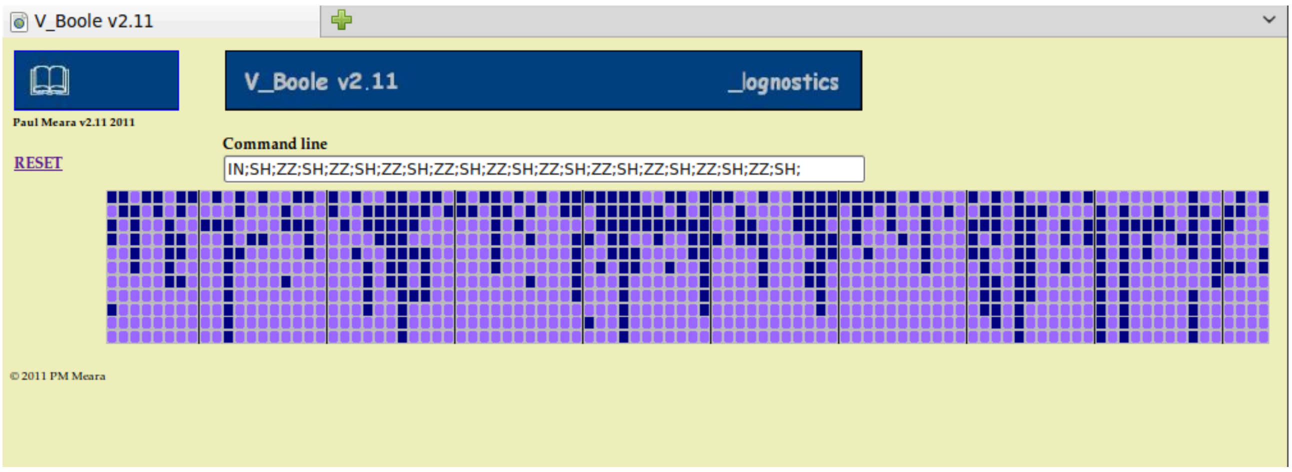
You should have a single line containing 100 squares of different colours. Each square represents one node in the network, and the colours tell you the current state of the nodes. Dark nodes are ON, light-coloured nodes are OFF. You should find that about half the nodes are ON and half are OFF after initialisation.

5: Now go back to the start page. You can do this by clicking on **Reset**, or by using your browser's BACK button.

6: The start page contains a list of commands that you can use to work on your network. The simplest command is **ZZ**. This command tells the program to iterate the network: the program looks at al the nodes in the network, and computes a number of changes depending on how each node is affected by its inputs. Type the commands:

```
IN;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;
```

i.e. IN;H; followed by ten copies of **ZZ;SH;**. This instruction set will initialise the network and display it. Then it will update the network and display the results ten times. Click on the **SUBMIT** button. The resulting output looks like this:



Command line

```
IN;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;ZZ;SH;
```

Clearly, writing out instruction sets in detail like this is awkward, so V_Boole v2.11 lets you compress repeated instruction sets. This one can be rewritten as:

IN;SH;10(ZZ,SH);

Note that instructions inside parentheses are separated by commas, and not by semi-colons.

7: The next simple command is BL. This inserts a blank line in the output. It is useful the output is particularly complex, and you need to insert markers so that you can see where particular instructions kick in.

8: The remaining instructions are more complex since they come with an additional numerical parameter that follows the alphabetical instruction code. These instructions work on individual nodes, rather than on the network as a whole.

ON activates nodes
OF deactivates nodes
LA changes a node's first link by assigning another random node to this role.
LB changes a node's second link by assigning another random node to this role.
RT makes a node more difficult to activate.
LT makes a node easier to activate.
FL flips the current value of a node. An ON item goes OFF and an OFF item goes ON.

The numerical parameter that accompanies each instruction is the probability that the instruction applies to any individual node. So,

ON10; means that there is a 10% chance that any node will be activated.
FL5; means that there is a 5% chance that any node will have its current value flipped.
RT1; means that there is a 1% that a node will become harder to activate.
OF100; will deactivate all the nodes.
LA100;LB100; completely relinks all the nodes.

9: Since there are 100 nodes in the network, the numerical parameter corresponds roughly to the number of nodes that will be affected. For example,

FL10; will flip the value of about 10 nodes.
RT2; will make approximately two nodes more difficult to activate;
ON50; will activate about half the nodes.

10: Here are some examples of how these codes are used in practice.

IN;20(ZZ,SH);BL;FL100;SH;

Initialise, update 20 times, show the results after each update, insert a blank line, flip all the values and show the results.

IN;20(ZZ);SH;20(ON10,SH);

Initialise, update 20 times, show the last update; turn on about 10 items and display the results twenty times.

11: Note that the probabilistic nature of the parameters means that some of these codes have slightly unpredictable results. For example:

ON10;

will usually turn activate about 10 nodes, but if most of the nodes are already activated, then this command might not easily find nodes to activate. You can check how many nodes this instruction activates at any time by using the instruction:

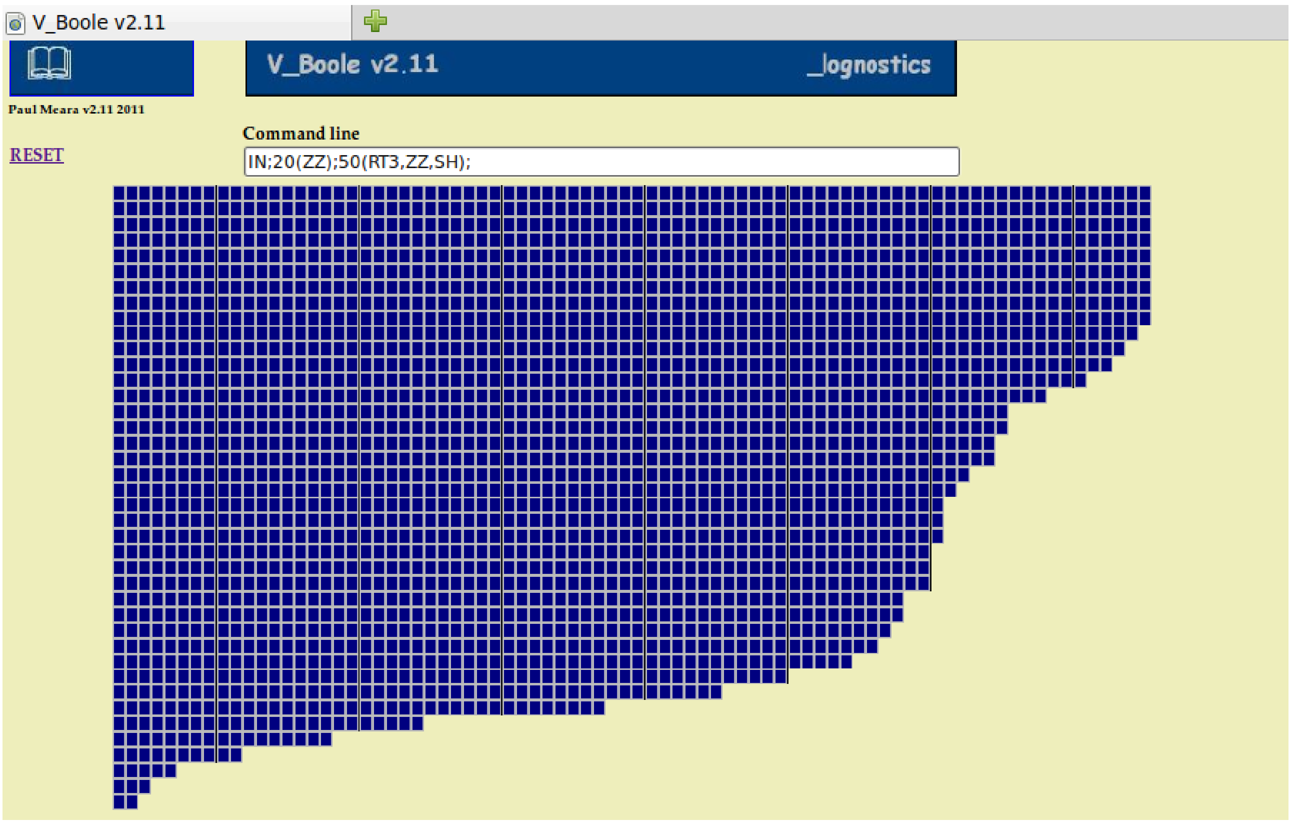
SH;ON10;ZZ;SH;

This instruction will display the network, make the designated changes, update the network and display the new state of the network.

12: Note that

10(ON1,ZZ); SH; will not give you the same result as ON10;ZZ;SH;

13: V_Boole v2.11 gives you a limited amount of control over the way the program displays the data. The Display Mode can be changed by altering the value in the Md Box. If this value is set to 1, then all nodes will be displayed, irrespective of whether they are activated or not. If this value s set to 0 then only activated nodes are displayed. This facility is useful if you want to count the overall number of activated nodes, rather than the individual nodes that are activated. An example of this display mode is shown below.



IN;20(ZZ);50(RT3,ZZ,SH);

14: V_Boole v2.11 also gives you some control over the way the program deals with randomisation in the network.

The box marked RCode determines the randomisation pattern the program uses to initialise the network. The box marked RSeq determines the randomisation pattern the program uses when it makes changes to a network. This allows you to test the same network in a number of different ways, or to apply a single set of changes to a number of different networks.

Part 2:

This section of the manual contains a number of suggestions for things you can experiment with. Some of these ideas have already been developed in the papers listed in the bibliography. For all the suggestions in this section, you should try out different values of RCode and RSeq. You should also change the numbers in command lines below.

1: Run IN;SH;20(ZZ,SH);

This set of instructions randomly initialises a network, and then iterates it twenty times.

What you should expect:

Boolean networks normally settle into a stable state where all or very nearly all the nodes do not change their level of activity. These steady states are called attractor states. A random initialisation with 100 nodes will normally reach an attractor state within 10 iterations.

2: Run: IN;20(ZZ);SH;20(ON1,ZZ,SH);20(ZZ);

This set of instructions randomly initialises a network and iterates it 20 times. It then displays the stable state. Next it turns one item on, iterates the network and shows the result. It repeats this last process 20 times, then allows the network to return to a stable state.

What you should expect:

Boolean networks are astonishingly stable, and small perturbations in the network will normally disappear immediately.

3: Run: IN;20(ZZ);SH;20(ON5,ZZ,SH); 20(ZZ);

This is the same as the previous instruction set, except that the number of items activated at each pass is 10 instead of 1.

What you should expect:

Changing a large number of items might destabilise the network. This pattern usually produces a very high level of activity in the network, but once the activation is stopped, the network will restabilise itself. Changes as large as this sometimes result in the network stabilising in a different state.

4: Run: IN;20(ZZ);SH;20(OF5,ZZ,SH); 20(ZZ);

This is the same pattern of instructions as No 3, except that here we are deactivating items, rather than activating them.

What you should expect:

Activation levels will go down, rather than up. Once the deactivation stops, then the network should return to its attractor state.

5: Run: IN;20(ZZ);SH;20(FL5,ZZ,SH);20(ZZ);

This is the same pattern of instructions as No 3, except that items can be activated or deactivated depending on their current state.

What you should expect:

As before, the network will usually return to its original attractor state once the value-flipping tops.

6: Run: IN;20(ZZ,SH);OF100;SH;ON10;20(ZZ,SH);

This set of instructions initialises and stabilises a network. Then it deactivates all the items. It follows this by activating a small number of items, and allowing the network to find an attractor state.

What you should expect:

Usually the network restores itself to its original attractor state, but sometimes it settles in a new attractor state that is slightly less active than the original one.

7: Run: IN;20(ZZ,SH);OF100;SH;30(ON2,ZZ,SH);20(ZZ,SH);

A slightly different way of demonstrating the idea that deactivated networks retain a memory of what they should be. Here we have a series of small kicks instead of one large one.

What you should expect:

The higher the number of items that are activated, the more likely it is that the network will return to its attractor state.

8: Run: IN;20(ZZ,SH);50(TU1,ZZ,SH);

This set of instructions is the basic model for investigating attrition in networks. The instructions set up and stabilise a network. Then at each iteration of the model, one item is made harder to activate.

What you should expect:

Usually this set of instructions shows no effect at all on the network for twenty or thirty iterations. Then without warning, the entire system suddenly collapses.

Some further questions:

All the examples above appear to have analogues with the behaviour of real lexicons. See if you can think of other examples which would allow you to investigate features of real lexicons which we have not included here.

V_Boole v2.11 is a small scale simulation which allows only 100 nodes in the network, so you can't use it to explore what happens in large vocabularies. It is possible that very large networks have properties which are rather different from those of a smaller, densely connected one. The other main shortcoming of V_Boole v2.11 is that it starts out with an established network. You can explore what happens when you restructure the network, using the LA and LB commands, but you cannot examine how a Boolean network grows from scratch. We might expect that a network that starts out small and gets bigger would behave very differently from this preliminary static model. These ideas will be explored in later versions of V_Boole.

Part 3:

The technical bits

V_Boole v2.11 is written in HTML and PERL. The underlying PERL script is available to bona fide researchers on request. Contact the author if you would like a copy.

The programs should work on any up to date browser. If you use a browser that produces unusual output, please let us know,

Improvements and suggestions for new features

Suggestions for improvements and new features are always welcome. Contact the author.

Citations
Please acknowledge this program if you use it in your own work. The preferred citation format is:
PM Meara
V_Boole v2.11 Swansea: Lognostics. 2011. available at http://www.lognostics.co.uk/tools/

Contact us
Paul Meara can be contacted on **p.m.meara@gmail.com**

Part 4:
Bibliography

Meara, PM
Self-organisation in bilingual lexicons. In: **P Broeder and J Muure** (Eds.) *Language and Thought in Development*. Tubingen: Narr. 1999. 127-144.
Meara, PM
Simulating recovery from bilingual aphasia. *International Journal of Bilingualism* 3,1(1999), 45-54.
Meara, PM
Modelling attrition in vocabularies. In: **A Hauksdottir, B Arnbjornsdottir, M Gardarsdottir and S Þorvaldsdottir** (Eds.) *Forskning I Nordiske Sprog som andet- og fremmedsprog.* Reykjavik: Haskoli Islands – Haskolautgatan. 2002. 153-175.
Meara, PM
Modelling vocabulary loss. *Applied Linguistics* 25,2(2004), 137-155.
Meara, PM
Emergent properties of multilingual lexicons. *Applied Linguistics* 27,4(2006), 620-644.
Meara, PM
Growing a vocabulary. *EuroSLA YearBook* 7(2007), 49-65.
Meara, PM *in preparation*
Boolean Lexicons. Swansea: Lognostics. 2011