How to Find Control Targets

This document contains step-by-step directions for implementing the control techniques in Thesis chapter 4 titled *Phenotype Control Theory*.

Written by Daniel Plaugher, 9/14/22

Computational Algebra

1. Navigate to http://www2.macaulay2.com/Macaulay2/ .
    a. To become more acquainted with the M2 language, browse the various documentations and tutorials.
2. On the left, select 'Macaulay2Web' under "Try It Out".
    a. Code can simply be copied and pasted into the platform.

Implementation of ***node*** control

1. Execute the first section defining variables and rings.
2. Run the section for defining the BN and finding fixed points.
3. If there are desired limits to which variables to consider, define which UP/UM variables to remove (as seen in thesis Eq 4.10). Otherwise, simply define the BN with control parameters as 'Fcontrol'.
    a. To only consider strictly UP or UM (i.e knock-in or knockout), run the appropriate sections under "ONLY KNOCK-IN" or "ONLY KNOCKOUT".
4. Choose control objectives (region blocking vs. new fixed point) and define appropriate parameters.
    a. There are separate sections for each objective. For example, use region blocking for thesis Eq. 4.10 vs. new fixed point creation for thesis Eq. 4.7.
5. If needed, use the final section to parse the generators that only contain parameters.

Implementation of ***edge*** control

1. Execute the first section defining variables.
2. If there are desired edge limitations (as in thesis Eq. 4.6), use the next section to define such limitations. Otherwise, let PARSselected=PARSstring.
3. Define the ring.
4. Run the section for defining the BN and finding fixed points.
5. Encode the edge controls according to the search for knockouts or constant expression.
6. Choose control objectives (transition blocking vs. new fixed point) and define appropriate parameters.
    a. There are separate sections for each objective. For example, use transition blocking for thesis Eq. 4.8 vs. new fixed point creation for thesis Eq. 4.6.
7. If needed, use the final section to parse the generators that only contain parameters.

The outputs for each of the objectives will match those listed under the examples in thesis section 4.1 *Algebraic Methods (CA)*

Control Kernel

1. Download the appropriate packages from https://doi.org/10.5281/zenodo.5172898.
2. Within the 'control_kernel_code' file, open the python file 'control_kernel_analysis'.
3. Create a separate text file defining your Boolean network, this must be saved under

    '*Users/…control_kernel_code/neet/boolean/data/file_name.txt*'

4. Scroll to the bottom of the python file and call:

```
#%%
from neet.boolean import LogicNetwork
net = LogicNetwork.read_logic('C:/Users/…/control_kernel_code/neet/boolean/data/file.txt')
print(net.names)
ck=ck_analysis(net)
print(ck)
```

It is important to note that the lexicographical output for fixed points is in reverse order. So instead of (1,…,n) it becomes (n,…,1). For the example in thesis section 4.2, the output should be:

> Finding attractors and control kernels...
> {'size': 6, 'control_kernel_sizes': [1, 1], 'control_kernels': [{0}, {0}], 'attractors': [[8], [55]], 'has_limit_cycles': False, 'delta_control_nodes': {0: [{0}, {0}]}, 'basin_entropies': {0: [0.11611507530476972, 0.11611507530476972]}, 'delta_control_kernel_sizes': array([1, 1]), 'delta_basin_entropies': array([0.11611508, 0.11611508]), 'simple_control_entropies': False, 'control_kernel_time_minutes': 0.00026041666666666666, 'modules': [{0, 1, 2, 3, 4, 5}]}

Feedback Vertex Set

1. Download the appropriate packages from https://github.com/jgtz/FVS_python3 .
2. Open and run the 'FVS_example.py' file.

Note that the code only outputs a single FVS, not every FVS. To see more FVSs, simply re-run the code.

For the example in thesis section 4.3, the output should be:

> ['x2'], ['x3'], ['x4'], which does not include the input node x1

Modularity via Simple networks

1. Open the MATLAB file 'thesis_examples.m'.
2. Add the appropriate paths (listed at the top of the file).
3. Scroll down to the section titled *Modularity Example* and run the section .

The output will be a di-graph showing the condensation graph (module hierarchy). To determine the modules and their elements, use variables *bin* and *binsize*. This particular example has two bins, each with 3 elements. The variable *modTable* contains column-wise storage of the modules with their corresponding nodes.

Stable Motifs

1. Download the appropriate files from (a) https://github.com/jgtz/StableMotifs and/or (b) https://github.com/jcrozum/pystablemotifs. The following steps will depend on which version you use.

If using (a)

2. Save the Boolean rules file 'SM_example.txt' in the '*dist'* subfolder within the '*StableMotifs-master'* folder.
3. To run the project, go to the command line terminal, navigate to the folder where the 'StableMotifs.jar' file and the 'lib' folder are located.
4. Type the command: *java -jar StableMotifs.jar SM_example.txt*

The output will contain the following:

- 2 tab separated TXT files with the quasi-attractors.
- A folder with the reduced networks for the first tab separated TXT file.
- A folder with the reduced networks for the second tab separated TXT file.
- A tab separated TXT file with the stable motifs found during network reduction.
- 2 tab separated TXT file with the sequences of stable motifs composing the stable motif succession diagram, and each transition in the diagram.
- A TXT file with the stable motif control sets.

For our example, the most important takeaways are the quasi-attractors:

x1 x2 x3

1 0 1

0 1 1

And the control sets

x2=0 Attractor0

x3=1 Attractor0

x2=1 Attractor1

x3=0 x1=1 Attractor1

Now, if using (b), there are two options

1. To use the system command prompt, navigate the directory to the ‘pystablemotifs-master’ folder.
2. Save the Boolean rules file ‘SM_example.txt’ within the ‘models’ subfolder of ‘pystablemotifs-maser’
3. In the command line, execute “ python -m pystablemotifs "./models/SM_example.txt" ”

The output for our example contains

```
RULES
x1* = x3 | x2
x2* = x1 & !x3
x3* = !x2 | !x1

Analyzing network . . .
Analysis complete.

There are 2 attractors.
{'x1': 1, 'x2': 0, 'x3': 1}
{'x1': 1, 'x2': 1, 'x3': 0}
----------
Computing attractor control . . .
----------
Target: {'x1': 1, 'x2': 0, 'x3': 1}
```

Interventions:

{'x3': 1}

{'x2': 0}

Target: {'x1': 1, 'x2': 1, 'x3': 0}

Interventions:

{'x2': 1}

{'x1': 1, 'x3': 0}

Complete.

Lastly, to use the python file

1. Save the 'SM_example.py' file within the 'pystablemotifs-master' folder.
2. Import Boolean rules, either by directly as a string or as a separate file import.
3. Define the target(s) of desired node states.
4. Execute the file, which uses all search types, but individual methods can be used.
    a. For a great tutorial, view the 'Control Tutorial.ipynb' file within the "Examples and Tutorials" folder.

The output for our example is:

RULES

x1* = x3 | x2

x2* = x1 & !x3

x3* = !x2 | !x1

There are 2 attractors.

{'x1': 1, 'x2': 0, 'x3': 1}

{'x1': 1, 'x2': 1, 'x3': 0}

--- BRUTE FORCE ---

Time running method: 0.00013649999164044857

Sets found:

{'x3': 0}

{'x2': 1}

--- GRASP SEARCH ---

Time running method: 0.02848749999247957

Sets found:

{'x2': 1}

--- INTERNAL HISTORY ---

Time running method: 0.0001711999939288944

Sets found:

{'x2': 1}

{'x1': 1, 'x3': 0}

--- MINIMAL HISTORY ---

Time running method: 8.720000914763659e-05

Sets found:

One temporary intervention from each list, in order.

(1 interventions in total)

[{'x2': 1}]