iggy
User Guide
version 1.2

Sven Thiele

1 What is iggy?

iggy is a tool for consistency based analysis of influence graphs and observed systems behavior. For many biological systems knowledge bases are available that describe the interaction of its components usually in terms of causal networks and influence graphs. In particular signed influence graphs where edges indicate either positive or negative effect of one node upon another. Building upon a notion of consistency between biochemical/genetic regulations and high-throughput profiles of cell activity iggy implements methods to check the consistency of large-scale data sets and provides explanations for inconsistencies. In practice, this can be used to identify unreliable data or to indicate missing reactions. Further, iggy addresses the problem of repairing networks and corresponding yet often discrepant measurements in order to re-establish their mutual consistency and predict unobserved variations even under inconsistency

2 Prerequisites

iggy is a python application that uses the power of answer set solving technology to compute its solution. Therefore it depends on the PyASP library, a python wrapper for the solvers from the Potassco Answer Set Solving Collection. All dependencies are automatically resolved and installed via pip, the recommended package installer for the Python Package index where the software is hosted. The PyASP package detects the running system and installs the matching binaries of the answer set grounder gringo and solver clasp. iggy runs on the Linux and Mac OS operating systems Windows is currently **not** supported.

3 Installation

3.1 Installation using pip

You can install the iggy package by running:

```
$ pip install --user iggy
```

On Linux the executable scripts can then be found in /.local/bin
and on Mac OS the scripts are under /Users/YOURUSERNAME/Library/Python/3.1/bin.

3.2 Installation of pip

If pip is not installed one can install pip without admin rights. Therefore one has to first download getpip.py via:

```
$ wget https://raw.github.com/pypa/pip/master/contrib/get-pip.py
```

and then install pip locally:

```
$ python get-pip.py --user
```

Now it is possible to use the local pip and proceed with the section *Installation of pip*.

3.3 Installation without pip

Although we do not recommend it, it is possible to install iggy without pip. But then one has to take care of the dependencies oneself. Therefore one has to first download pyasp-1.4.1.

```
$ wget https://pypi.python.org/packages/source/p/pyasp/pyasp-1.4.1.tar.gz
```

Next one has to extract and install PyASP:

```
$ gzip -d pyasp-1.4.1.tar.gz
$ tar -xvf pyasp-1.4.1.tar
$ cd pyasp-1.4.1
$ python setup.py install --user
```

Then one needs to download `iggy-1.2`:

```
$ wget https://pypi.python.org/packages/source/i/iggy/iggy-1.2.tar.gz
```

and to extract and install ingranalyze:

```
$ gzip -d iggy-1.2.tar.gz
$ tar -xvf iggy-1.2.tar
$ cd iggy-1.2
$ python setup.py install --user
```

On Linux the executable scripts can then be found in `/.local/bin`
and on Mac OS the scripts are under `/Users/YOURUSERNAME/Library/Python/2.7/bin`.

4 Usage

Typical usage is:

```
$ iggy.py network.sif observation.obs --show_labelings 10 --show_predictions
```

For more options you can ask for help as follows:

```
$ iggy.py -h
usage: iggy.py [-h] [--no_zero_constraints]
           [--propagate_unambigious_influences] [--no_founded_constraint]
           [--autoinputs] [--scenfit] [--show_labelings SHOW_LABELINGS]
           [--show_predictions]
           networkfile observationfile

positional arguments:
  networkfile           influence graph in SIF format
  observationfile       observations in bioquali format

optional arguments:
  -h, --help            show this help message and exit
  --no_zero_constraints
           turn constraints on zero variations OFF, default is ON
  --propagate_unambigious_influences
           turn constraints ON that if all predecessor of a node
           have the same influence this must have an effect,
           default is ON
  --no_founded_constraint
           turn constraints OFF that every variation must be
           explained by an input, default is ON
  --autoinputs          compute possible inputs of the network (nodes with
           indegree 0)
  --scenfit             compute scenfit of the data, default is mcos
  --show_labelings SHOW_LABELINGS
           number of labelings to print, default is OFF, 0=all
  --show_predictions    show predictions
```

5 Input

`iggy` works with two kinds of data. The first is network data representing and influence graph model. The second is the experimental data, representing experimental condition and observed behavior.

5.1 Network data

The network data is represented as file in simple interaction format `SIF` as shown below. Lines in the `SIF` file specify a source node, a relationship type (or edge type), and one target node. For our influence graph models we have the edge types `1` for *increases* and `-1` for *decreases*. The first line in the example below therefore states that `depor` has an *activating* (`1`) influence on `shp2_ph`. The last line states that `shp2_ph` has a *inhibiting* (`-1`) influence on `plcg`. Duplicate entries are ignored. Multiple edges between the same nodes must have different edge types. Other edge types than `1` and `-1` will lead to a parsing error.

```
1 depor_p       1         shp2_ph
2 gab1_mem_p    1         shp2_ph
3 shp2_ph       -1        pi3k
4 gab1_mem_p    1         pi3k
5 pi3k          1         gab1_mem_p
6 shp2_ph       -1        plcg
```

5.2 Experimental data

The experimental data is given in the file format shown below. Nodes which are perturbed in the experimental condition are denoted as `input`. The first line of the example below states that `depor` has been perturbed in the experiment. This means `depor` has been under the control of the experimentalist and its behavior must therefore not be explained. The behavior of a node can be either `+`, `-`, `0`, `notPlus`, `notMinus`. Line 2 states that an *increase* (`+`) was obeserved in `depor`, as it is declared an `input` this behavior has been caused by the experimentalist. Line 3 states that `stat5ab_py` has *decreased* (`-`) and line 4 states that `ras` has *not changed* (`0`). Line 5 states that an *uncertain decrease* (`notPlus`) has been observed in `plcg` and line 6 states that an *uncertain increase* (`notMinus`) has been observed in `mtorc1`. Line 7 states that `akt` is initially on the minimum level, this means it cannot further decrease, and line 8 states that `gab1` is initially on the maximum level, this means it cannot further increase.

```
1 depor          = input
2 depor          = +
3 stat5ab_py     = -
4 ras            = 0
5 plcg           = notPlus
6 mtorc1         = notMinus
7 akt            = MIN
8 gab1           = MAX
```

6 Output

`iggy` presents the results of its analysis as text output. The output of `iggy` can be redirected into a file using the `>` operator. For example to write the results shown below into the file `myfile.txt` type:

```
$ iggy.py network.sif observation.obs --show_labelings 10 --show_predictions > myfile.txt
```

In the following we will dissect the output generated by `iggy`. The first 3 lines of the output state the constraints that have been used to analyze network and data. For our example it is the default setting with the following constraints. For a deeper understanding of these constraints see [2].

```
1 all observed changes must be explained by an predecessor
2 no-change observations must be explained
3 all observed changes must be explained by an input
```

Next follow some statistics on the input data. Line 4-5 tells us that the influence graph model given as `network.sif` consists of `96` nodes, with `116` edges with activating influence and `16` edges with inhibiting influence and `0` edges with `Dual` or ambiguous influence. Line 9 tells that the experimental data given as `observation.obs` in itself is `consistent`, which means it does not contain contradictory observations. Line 11 tells that the experimental conditions consists of `14` perturbations marked as `input` nodes, that `12` nodes were observed as increased `+`, `10` nodes *decreased* (`-`), `20` nodes did *not change* (`0`), `5` nodes were observed with an *uncertain decrease* (`notPlus`), `4` nodes were observed with an *uncertain increase* (`notMinus`), `74` nodes were `unobserved` and the experimental data contained `0` observations of things that are not in the given influence graph model.

```
4  Reading network network.sif ... done.
5     Nodes: 96  Activations: 116  Inhibitions: 16  Dual: 0
6
7  Reading observations observation.obs ... done.
8
9  Checking observations observation.obs ... done.
10    Observations are consistent.
11    inputs: 14  observed +: 12  observed -: 10  observed 0: 20  observed notPlus: 5
12    observed notMinus: 4 unobserved: 74  not in model: 0
```

Then follow the results of the consistency analysis. Line 14 tells us that network and data are inconsistent and that the size of a *minimal correction set* (`mcos`) is `1`. This means that at least `1` influence needs to be added to restore consistency. For a deeper understanding of mcos see [1]. Further the output contains at most `10` consistent labeling including correction set. This is because we choose to set the flag `--show_labelings 10`. In our example we have only `2` possible labelings. Each labeling represents a consistent behavior of the model (given mcos the corrections). `Labeling 1`, tells it is possible that `STAT3_n` and `PAK1` *increase* (`+`), `IGF1_act` does *not change* (`0`) and that `KS6A5/KS6A4` and `TNR1A/TNR1B` *decrease* (`-`). Line 26 tells us that this is a consistent behavior if `MTOR` would receive a positive influence, which is currently not included in the model. `Labeling 2`, represents an alternative behavior, here `PAK1` and `KS6A5/KS6A4` do *not change* (`0`). Please note that in this example both labelings are consistent under the same correction set. In another example more than one minimal correction set could exists.

```
13 Computing mcos of network and data ... done.
14    The network and data are inconsistent: mcos = 1.
```

```

Compute mcos labelings ... done.
Labeling 1:
   gen("STAT3_n") = +
   gen("PAK1") = +
   gen("IGF1_act") = 0
   gen("KS6A5/KS6A4") = -
   gen("TNR1A/TNR1B") = -

   labeled +: 2  labeled -: 2  labeled 0: 1

   new_influence("observation.obs",gen("MTOR"),1)

Labeling 2:
   gen("STAT3_n") = +
   gen("PAK1") = 0
   gen("IGF1_act") = 0
   gen("KS6A5/KS6A4") = 0
   gen("TNR1A/TNR1B") = -

   labeled +: 1  labeled -: 1  labeled 0: 3

   new_influence("observation.obs",gen("MTOR"),1)
```

Finally the prediction results are listed. A prediction is a statement that hold under all labeling under all minimal repairs. For a formal definition of predictions see [2]. Here the predictions say that `STAT3_n` *always increases* (`+`), `PAK1` *never decreases* (`NOT -`), `IGF1_act` always stays *unchanged* (`0`), `KS6A5/KS6A4` *never increases* (`NOT +`), and that `TNR1A/TNR1B` *always decreases* (`-`).

```
Compute predictions under mcos ... done.
   gen("STAT3_n") = +
   gen("PAK1") = NOT -
   gen("IGF1_act") = 0
   gen("KS6A5/KS6A4") = NOT +
   gen("TNR1A/TNR1B") = -

   predicted +: 1  predicted -: 1  predicted 0: 1  predicted NOT +: 1
   predicted NOT -: 1  predicted CHANGE: 0
```

References

[1] Ioannis N. Melas, Regina Samaga, Leonidas G. Alexopoulos, and Steffen Klamt. Detecting and Removing Inconsistencies between Experimental Data and Signaling Network Topologies Using Integer Linear Programming on Interaction Graphs. *PLoS Computational Biology*, 9(9):e1003204, 09 2013.

[2] Sven Thiele, Luca Cerone, Julio Saez-Rodriguez, Anne Siegel, Carito Guziolowski, and Steffen Klamt. Extended notions of sign consistency to relate experimental data to regulatory network topologies. 2015.