Iggy - User Guide (version 2.0.0)

Sven Thiele

What are `iggy` and `opt_graph` ?

`iggy` and `optgraph` are tools for consistency based analysis of influence graphs and observed systems behavior (signed changes between two measured states). For many (biological) systems are knowledge bases available that describe the interaction of its components in terms of causal networks, boolean networks and influence graphs where edges indicate either positive or negative effect of one node upon another.

`iggy` implements methods to check the consistency of large-scale data sets and provides explanations for inconsistencies. In practice, this is 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.

`opt_graph` confronts interaction graph models with observed systems behavior from multiple experiments. `opt_graph` computes networks fitting the observation data by removing (or adding) a minimal number of edges in the given network.

You can download the precompiled binaries for 64bit linux and macos on the release page.

Sample data is available here: demo_data.tar.gz

Compile yourself

Clone the git repository:

```
git clone https://github.com/bioasp/iggy.git
cargo build --release
```

The executables can be found under `./target/release/`

Input Model + Data

`iggy` and `opt_graph` work with two kinds of data. The first is representing an interaction graph model. The second is the experimental data, representing experimental condition and observed behavior.

Model

The model is represented as file in complex interaction format `CIF` as shown below. Lines in the `CIF` file specify a interaction between (multiple) source nodes and one target node.

```
shp2                        ->  grb2_sos
!mtor_inhibitor             ->  mtor
?jak2_p                     ->  stat5ab_py
!ras_gap&grb2_sos           ->  pi3k
akt&erk&mtor&pi3k           ->  mtorc1
gab1_bras_py                ->  ras_gap
gab1_ps&jak2_p&pi3k         ->  gab1_bras_py
```

In our influence graph models we have simple interactions like: in Line 1 for `shp2` *increases* `grb2_sos` and in Line 2 the `!` indicates that `mtor_inhibitor` tends to *decrease* `mtor`. in Line 3 the `?` indicates an unknown influence of `jak2_p` on `stat5ab_py`. Complex interactions can be composed with the `&` operator to model a combined influence of multiple sources on a tartget. In Line 4 an decrease in `ras_gap` with an increase in `grb2_sos` tend to increase `pi3k`.

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 `grb2_sos` is initially on the maximum level, this means it cannot further increase.

```
depor          = input
depor          = +
stat5ab_py     = -
ras_gap        = 0
jak2_p         = notPlus
mtorc1         = notMinus
akt            = MIN
pi3k           = MAX
```

Iggy

`iggy` performs consistency checks for an interaction model and a data profile. It computes explanations (minimal inconsistent cores *mics*) for inconsistencies and suggests repairs for model and data.

The *mics* are connected parts of the model and indicate unreliable data or missing reactions. The repairs re-establish the mutual consistency between model and data, and enable predictions of unobserved behavior even under inconsistency.

The typical usage of `iggy` is:

```
$ iggy -n network.cif -o observation.obs -l 10 -p
```

For more options you can ask for help as follows:

```
$ iggy -h
iggy 2.0.0
Sven Thiele <sthiele78@gmail.com>
Iggy confronts interaction graph models with observations of (signed) changes between
two measured states (including uncertain observations). Iggy discovers inconsistencies
in networks or data, applies minimal repairs, and predicts the behavior for the
unmeasured species. It distinguishes strong predictions (e.g. increase in a node) and
weak predictions (e.g., the value of a node increases or remains unchanged

USAGE:
    iggy [FLAGS] [OPTIONS] --network <networkfile>

FLAGS:
    -a, --autoinputs                Declare nodes with indegree 0 as inputs
        --depmat                    Combine multiple states, a change must be explained
                                    by an elementary path from an input
        --elempath                  Every change must be explained by an elementary
                                    path from an input
        --founded_constraints_off   Disable foundedness constraints
        --fwd_propagation_off       Disable forward propagation constraints
    -h, --help                      Prints help information
        --mics                      Compute minimal inconsistent cores
        --scenfit                   Compute scenfit of the data, default is mcos
    -p, --show_predictions          Show predictions
    -V, --version                   Prints version information
OPTIONS:
    -l, --show_labelings <max_labelings>   Show max_labelings labelings, default is OFF,
                                            0=all
    -n, --network <networkfile>            Influence graph in CIF format
    -o, --observations <observationfile>   Observations in bioquali format
```

Compute *mcos* or *scenfit* and predictions under inconsistency

`iggy` implements two measures for inconsistency *mcos* and *scenfit*. While *mcos* measures the minimal number of observations that cannot be explained, *scenfit* measures a minimal number of changes to the model to re-establish the mutual consistency between model and data. The default in iggy is *mcos* but *scenfit* can be used with the option `--scenfit`.

With the option `--show_labelings, -l N` `iggy` computes at most `N` such labelings and repairs that are consistent.

With the flag `--show_predictions, -p` `iggy` computes predictions under inconsistencies. More precisely the behaviors of the system that are invariant also under the minimal repairs.

`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 -n network.cif -o observations.obs -l 10 -p > 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~\cite{sthiele15}.

```
+ All observed changes must be explained by an predecessor.
+ 0-change must be explained.
+ 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.cif` consists of `18` species nodes and `4` complex nodes, with `19` edges with activating influence and `6` edges with inhibiting influence and `1` edge with `Unknown` 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 has `2` perturbations marked as `input` nodes, that `4` nodes were observed as increased `+`, `1` node *decreased* (`-`), `7` nodes did *not change* (`0`), `1` node were observed with an *uncertain decrease* (`notPlus`), `1` node were observed with an *uncertain increase* (`notMinus`), `1` node were observed with an *minimum level* (`MIN`), `1` node were observed with an *maximum level* (`MAX`), `4` nodes were `unobserved` and the experimental data contained `0` observations of species that are not in the given model.

```
Reading network model from "network.cif".

# Network statistics

    OR nodes (species): 18
    AND nodes (complex regulation): 4
    Activations = 19
    Inhibitions = 6
    Unknowns = 1

Reading observations from "observations.obs".

# Observations statistics

    unobserved nodes     : 4
    observed nodes       : 18
     inputs                : 2
     +                     : 4
     -                     : 1
     0                     : 7
     notPlus               : 1
     notMinus              : 1
     Min                   : 1
     Max                   : 1
     observed 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 `2` . This means that at least `2` influences need to be added to restore consistency. For a deeper understanding of mcos see~\cite{samaga13a}. 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 `4` possible labelings. Each labeling represents a consistent behavior of the model (given mcos the corrections). `Labeling 1` , tells it is possible that `mek1` *increases* (`+`), `shp2_ph` and `mtorc` do *not change* (`0`) and that `stat5ab_py` *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 3` , represents an alternative behavior, here `mtorc1` does *increases* (`+` }). 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.

```
The network and data are inconsistent: mcos = 2.

Compute mcos labelings ... done.
Labeling 1:
    mtorc1 = 0
    ras_gap = 0
    shp2 = 0
    gab1_bras_py = 0
    jak2_p = 0
    mek1 = +
    erk = +
    brb2 = 0
    akt = 0
    stat5ab_py = -
    brb = -
    gab1_ps = +
    grb2_sos = 0
    socs1 = 0
    pi3k = 0
    mtor = 0
    mtor_inhibitor = 0
    depor = +

 Repairs:

Labeling 2:
    mtorc1 = 0
    ras_gap = 0
    shp2 = 0
    gab1_bras_py = 0
    jak2_p = 0
    mek1 = +
    erk = +
    brb2 = 0
    akt = 0
    stat5ab_py = -
    brb = +
    gab1_ps = +
    grb2_sos = 0
    socs1 = 0
    pi3k = 0
    mtor = 0
    mtor_inhibitor = 0
    depor = +

 Repairs:

Labeling 3:
    mtorc1 = +
    ras_gap = 0
```

```
    shp2 = 0
    gab1_bras_py = 0
    jak2_p = 0
    mek1 = +
    erk = +
    brb2 = -
    akt = 0
    stat5ab_py = -
    brb = -
    gab1_ps = +
    grb2_sos = 0
    socs1 = 0
    pi3k = 0
    mtor = 0
    mtor_inhibitor = 0
    depor = +

  Repairs:

 Labeling 4:
    mtorc1 = +
    ras_gap = 0
    shp2 = 0
    gab1_bras_py = 0
    jak2_p = 0
    mek1 = +
    erk = +
    brb2 = -
    akt = 0
    stat5ab_py = -
    brb = +
    gab1_ps = +
    grb2_sos = 0
    socs1 = 0
    pi3k = 0
    mtor = 0
    mtor_inhibitor = 0
    depor = +

  Repairs:
```

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~\cite{sthiele15}. Here the predictions say that `gab1_ps` *always increases* (`+`), `stat5ab_py` *always decreases* (`-`), `shp2` always stays *unchanged* (`0`), `mtorc1` *never decreases* (`notMinus`), and `brb2` always stays *never increases* (`notPlus`),

```
Compute predictions under mcos ... done.

# Predictions:

    mek1 = +
    erk = +
    gab1_ps = +
    depor = +
    stat5ab_py = -
    ras_gap = 0
    shp2 = 0
    gab1_bras_py = 0
    jak2_p = 0
    akt = 0
    grb2_sos = 0
    socs1 = 0
    pi3k = 0
    mtor = 0
    mtor_inhibitor = 0
    brb2 = notPlus
    mtorc1 = notMinus
    brb = CHANGE

    predicted +        = 4
    predicted -        = 1
    predicted 0        = 10
    predicted notPlus  = 1
    predicted notMinus = 1
    predicted CHANGE   = 1
```

Compute minimal inconsistent cores `--mics`

Iggy computes minimal inconsistent cores *mics* for inconsistent model and data. The *mics* are connected parts of the model and indicate unreliable data or missing reactions. To compute the minimal inconsistent cores use the flag `--mics` as follows:

```
$ iggy -n data/Yeast/yeast_guelzim.cif  -o data/Yeast/yeast_snf2.obs --mics
_____________________________________________________________________________

+ All observed changes must be explained by an predecessor.
+ 0-change must be explained.
+ All observed changes must be explained by an input.
_____________________________________________________________________________

Reading network model from "data/Yeast/yeast_guelzim.cif".

# Network statistics
OR nodes (species): 477
AND nodes (complex regulation): 0
Activations = 665
Inhibitions = 270
Unknowns = 0

Reading observations from "data/Yeast/yeast_snf2.obs".

# Observations statistics

    unobserved nodes     : 388
    observed nodes       : 574
     inputs                : 0
     +                     : 376
     -                     : 198
     0                     : 0
     notPlus               : 0
     notMinus              : 0
     Min                   : 0
     Max                   : 0
     observed not in model : 485

Computing mcos of network and data ... done.

The network and data are inconsistent: mcos = 530.

Computing minimal inconsistent cores (mic's) ... done.

mic 1:
    YAL063C YER065C

mic 2:
    YBR159W YNL009W

mic 3:
    YJL159W YGR108W

mic 4:
    YPR119W YGR108W
```

```
mic 5:
    YMR307W YIL013C

mic 6:
    YNL241C YLR109W

mic 7:
    YOL006C YMR186W

mic 8:
    YGR108W YDR224C YAL040C

mic 9:
    YPL256C YIL072W YNL210W YGR044C YPR119W YJL194W YJL106W YDL179W YOR159C YHR055C YLR131C

mic 10:
    YPL256C YIL072W YNL210W YGR044C YJL194W YJL106W YDL179W YOR159C YHR055C YLR131C YDR522C

mic 11:
    YPL256C YIL072W YJL159W YNL210W YGR044C YJL194W YJL106W YDL179W YOR159C YHR055C YLR131C

mic 12:
    YPL256C YMR199W YIL072W YGL089C STA3 YLR452C YNL210W YIL099W YGR044C YIR019C YJL157C YBR

mic 13:
    YCR065W YOL116W YKR099W YGL073W YBR279W YGL025C YDR448W YDR392W YIR023W YLR451W YBR112C
```

Opt_graph

`opt_graph` confronts interaction graph models with observed systems behavior from multiple experiments. `opt_graph` computes networks fitting the observation data by removing (or adding) a minimal number of edges in the given network.

Typical usage is:

```
$ opt_graph -n network.cif -o observations_dir/ --show_repairs 10
```

For more options you can ask for help as follows:

```
$ opt_graph -h
opt_graph 2.0.0
Sven Thiele <sthiele78@gmail.com>
Opt-graph confronts interaction graph models with observations of (signed) changes between
two measured states. Opt-graph computes networks fitting the observation data by removing
(or adding) a minimal number of edges in the given network

USAGE:
    opt_graph [FLAGS] [OPTIONS] --network <networkfile> --observations <observationdir>
FLAGS:
    -a, --autoinputs               Declare nodes with indegree 0 as inputs
        --depmat                   Combine multiple states, a change must be explained by
                                   an elementary path from an input
        --elempath                 Every change must be explained by an elementary path
                                   from an input
        --founded_constraints_off  Disable foundedness constraints
        --fwd_propagation_off      Disable forward propagation constraints
    -h, --help                     Prints help information
    -V, --version                  Prints version information
OPTIONS:
    -r, --show_repairs <max_repairs>      Show max_repairs repairs, default is OFF, 0=all
    -n, --network <networkfile>           Influence graph in CIF format
    -o, --observations <observationdir>   Directory of observations in bioquali format
    -m, --repair_mode <repair_mode>       Repair mode: remove = remove edges (default),
                                                       optgraph = add + remove edges,
                                                       flip = flip direction of edges
```

Example

```
opt_graph -n in_silico_HEK293/v1_comp_BN.cif -o in_silico_HEK293/prior_data --depmat  -r 0 -
_____________________________________________________________________________

+ DepMat combines multiple states.
+ An elementary path from an input must exist to explain changes.
_____________________________________________________________________________

Reading network model from "data/in_silico_HEK293/v1_comp_BN.cif".

# Network statistics

    OR nodes (species): 23
    AND nodes (complex regulation): 12
    Activations : 47
    Inhibitions : 12
    Unknowns : 0

Reading observations from in_silico_HEK293/prior_data/first_response_mek1_up.txt.

Reading observations from in_silico_HEK293/prior_data/first_response_pi3k_down.txt.

Reading observations from in_silico_HEK293/prior_data/first_response_pi3k_mek1_down.txt.

Computing repair through add/removing edges ...
    using greedy method ... done.

The network and data can reach a scenfit of 0.

Repair 1:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: mek1 -> gab1_ps
    add edge: mek1 -> gab1_ps

Repair 2:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: !grb2_sos -> gab1_ps
    add edge: !grb2_sos -> gab1_ps

Repair 3:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: akt -> gab1_ps
    add edge: akt -> gab1_ps

Repair 4:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: mtorc2 -> gab1_ps
```

```
    add edge: mtorc2 -> gab1_ps

Repair 5:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: mtorc1 -> gab1_ps
    add edge: mtorc1 -> gab1_ps

Repair 6:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: erk -> gab1_ps
    add edge: erk -> gab1_ps

Repair 7:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: !gab1_bras_py -> gab1_ps
    add edge: !gab1_bras_py -> gab1_ps

Repair 8:
    remove edge: !mek1 -> shp2
    remove edge: mek1 -> stat5ab_py
    add edge: !ras_gap -> gab1_ps
    add edge: !ras_gap -> gab1_ps
```