Equibel Tutorial

Paul Vicol

May 29, 2015

1 Installation

Equibel currently supports Mac OS X and Linux, with Python 2.7.x (tested on 2.7.6). It is highly recommended that you install Equibel in a *virtual environment*. Virtual environments provide isolation between different Python projects, allowing you to have separate installations of Python for each project. To create a virtual environment, you must first install `virtualenv`:

```
$ sudo pip install virtualenv
```

Once `virtualenv` is installed, you can create a project directory and initialize a virtual environment in it as follows:

```
$ mkdir equibel_projects
$ cd equibel_projects
$ virtualenv venv --python=python2.7
```

The last line above creates a directory called `venv` which contains an installation of Python 2.7 that is isolated from the global system installation. A virtual environment provides you with an isolated space in which you can install specific versions of Python packages needed for a project, such that they do not interfere with different versions of the same packages installed system-wide or in other virtual environments. Note that we included the option `--python=python2.7` to ensure that Python 2.7 is used; this is necessary because Equibel is currently not compatible with Python 3.

Before you can install packages into the virtual environment, you have to *activate* it using

```
$ . venv/bin/activate
```

or

```
$ source venv/bin/activate
```

When you do this, your terminal prompt will update so that it is prepended by `(venv)`. Whenever you want to exit the virtual environment and return to using the system-wide Python installation, simply use:

```
$ deactivate
```

You can now install Equibel into the virtual environment as follows.

1. Get the source code by cloning the GitHub repository:

```
$ git clone git://github.com/asteroidhouse/equibel.git
```

2. Enter the equibel directory and run `pip`:

```
$ cd equibel
$ pip install .
```

This will install Equibel and its dependencies into `venv`. Equibel consists of two main components: the `equibel` package and the `equibeli` interactive command-line interface. The `equibel` package provides an API to the system, while the `equibeli` CLI allows for real-time experimentation with the system. The easiest way to get started using Equibel is to launch the CLI.

2 Using the `equibeli` CLI

2.1 Quickstart

Note: The virtual environment in which you installed Equibel must be activated for this to work.

To launch the CLI, simply type `equibeli` at the terminal:

```
$ equibeli
Equibel version 0.8.5
equibel (g) >
```

`equibeli` is structured around the idea of working with graphs: it provides commands to build graphs, assign formulas to nodes, perform belief change operations, and query nodes. The following is an example of an interactive session. Commands are entered at the `equibel (g) >` prompt, and responses from the system are indented.

```
equibel (g) > add_nodes [1,2,3,4]

    nodes: [1, 2, 3, 4]

equibel (g) > add_edges [(1,2), (2,3), (3,4)]

    edges:
        1 <-> 2
        2 <-> 3
        3 <-> 4

equibel (g) > add_formula 1 p&q

    node 1:
        q & p

equibel (g) > add_formula 4 ~p

    node 4:
        ~p
```

```
equibel (g) > one_shot

    One-shot belief change completed:
    ---------------------------------

    node 1:
        q & p
    node 2:
        q
    node 3:
        q
    node 4:
        q & ~p

equibel (g) > store completion1.bcf

    Successfully saved graph g to completion1.bcf.
```

This example creates a path graph on 4 nodes, assigns formulas to the first and last nodes, and performs a one-shot belief sharing operation. The initial and resultant graphs are shown below:

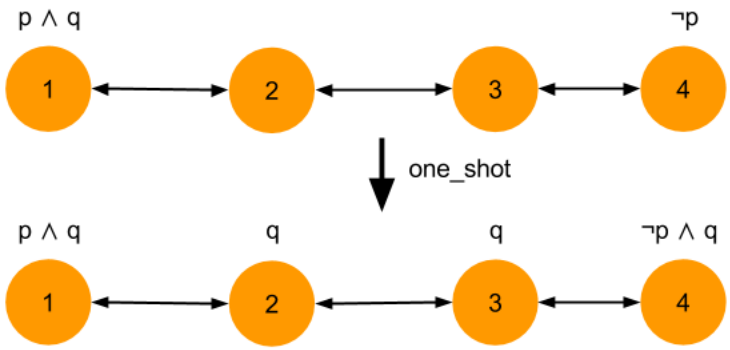


Figure 1: The completion of a path graph

2.2 Graph Contexts

In the CLI, every command is performed with respect to an *implicit graph*. The implicit graph is the *context* of the command. You always work with one graph at a time, and that graph is the current context. So, in the example above, the commands `add_nodes` and `add_edges` are really adding nodes and edges to the current context.

You can create new graphs, and you can have several graphs in one interactive session—you just have to switch contexts for your commands to affect another graph. In the prompt `equibel (g) >`, the part within the parentheses (i.e. `g`) is the name of the current context. The initial graph context for every interactive session is called `g`. You can list available graph contexts using the `graphs` command:

```
equibel (g) > graphs

    --g--
```

This will output the names of all graph contexts that have been created during the session; the name of *current* context is enclosed in `--` characters.

To create a new, blank-slate graph context, use the `create_graph` command:

```
equibel (g) > create_graph g2

    --g2--  g

equibel (g2) >
```

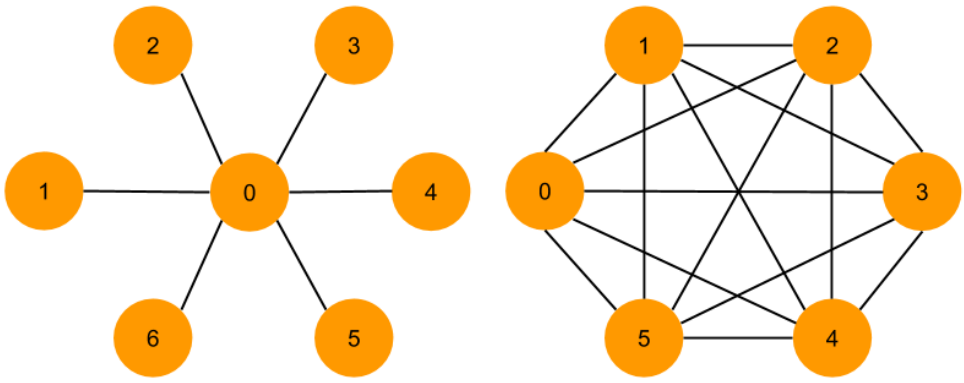
The argument to `create_graph` is the name of the new graph. Whenever you create a new graph, it automatically becomes the current context; as you can see above, the prompt updates to `equibel (g2) >`. Then, commands to add nodes, edges, formulas, etc. will be performed with respect to `g2`. To switch between different graph contexts, you can use the `use` command:

```
equibel (g2) > use g
equibel (g) >
```

As you can see, the prompt is updated to reflect the new graph context, `g`.

2.2.1 Creating Special Types of Graphs

The CLI also offers commands to create classic graph topologies, such as path graphs, star graphs, and complete graphs, automatically. Each of these commands has the format `create_[TYPE] GRAPH_NAME NUM_NODES`. Such a command creates a new graph context identified by `GRAPH_NAME`, and initializes the specified type of graph on `NUM_NODES` within the context. Note that whenever you use a generator function to create a graph on n nodes, the nodes are numbered from 0 to $n-1$.



(a) `create_star g2 6` (b) `create_complete g3 6`

- **Path Graphs**

  For example, the command `create_path g2 6` creates a new graph context named `g2`, and initializes a path graph on 6 nodes within that context:

  ```
  equibel (g) > create_path g2 6
  ```

```
    --g2--  g

equibel (g2) > nodes

    nodes: [0, 1, 2, 3, 4, 5]

equibel (g2) > edges

    edges:
        0 <-> 1
        1 <-> 2
        2 <-> 3
        3 <-> 4
        4 <-> 5
```

- **Star Graphs**

```
equibel (g2) > create_star g3 6

    --g3--  g2  g

equibel (g3) > nodes

    nodes: [0, 1, 2, 3, 4, 5, 6]

equibel (g3) > edges

    edges:
        0 <-> 1
        0 <-> 2
        0 <-> 3
        0 <-> 4
        0 <-> 5
        0 <-> 6
```

- **Complete Graphs**

```
equibel (g3) > create_complete g4 6

    --g4--  g3  g2  g

equibel (g4) > nodes

    nodes: [0, 1, 2, 3, 4, 5]

equibel (g4) > edges

    edges:
        0 <-> 1
        0 <-> 2
        0 <-> 3
        0 <-> 4
```

```
        0 <-> 5
        1 <-> 2
        1 <-> 3
        1 <-> 4
        1 <-> 5
        2 <-> 3
        2 <-> 4
        2 <-> 5
        3 <-> 4
        3 <-> 5
        4 <-> 5
```

2.3 CLI Help

To see a list of available commands, type `help`:

```
equibel (g) > help

Documented commands (type help <topic>):
========================================
add_atom   add_formula  atoms         edges     load         remove_edge   use
add_atoms  add_node     create_graph  formulas  nodes        remove_node
add_edge   add_nodes    create_path   graphs    quit         remove_nodes
add_edges  asp          directed      help      remove_atom  undirected

Undocumented commands:
======================
add_weight   containment  iterate   remove_formula  store
cardinality  e_iterate    one_shot  shell           weights
```

To get more details about a specific command, including a usage example, type `help <command>`:

```
equibel (g) > help add_edges

        Usage: add_edges EDGE_LIST

        Adds all the edges in the given list to the edges set.

        Example: add_edges [(1,2), (2,3), (3,4)]
```

3 Using the `equibel` API

You can use Equibel in Python programs by importing the `equibel` module. The following Python script can be found in `equibel/examples/api/manual_graph_creation.py`.

```
import equibel as eb

if __name__ == '__main__':
    G = eb.EquibelGraph()
```

```
    # Create edges. The nodes corresponding to the endpoints of
    # the edges are added automatically if not already present:
    G.add_edges([(1,2), (1,3), (3,4), (2,4)])

    # Add formulas to nodes:
    G.add_formula(1, "p")
    G.add_formula(2, "~p & q")
    G.add_formula(3, "r")

    # Find the completion of the G-scenario:
    R = eb.completion(G)

    # Print the resulting formulas at each node:
    for node_id in R.nodes():
        print("Node {0}: {1}".format(node_id, R.formulas(node_id)))
```

You can run this script as follows:

```
$ python manual_graph_creation.py
```

And the output should be:

```
Node 1: set([p & (q & p & r)])
Node 2: set([(q & ~p) & (~p & r)])
Node 3: set([q & r])
Node 4: set([true & (q & r)])
```

Note that the formulas are not simplified by default.

3.1 Graph Generators

`equibel` provides graph generators for several classic topologies, such as path, star, and complete graphs, as well as for well-known stochastic topologies, such as Waxman, Erdos-Renyi, and Barabasi-Albert graphs.