confidence-sample-hypothesis

January 10, 2017

```
In [4]: %load_ext autoreload
        %autoreload 2

In [5]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1 Confidence in hypothesis

This notebook demonsrtrates the evolution of confidence in a hypothesis. Depending on the hypothesis configuration of a subgraph we compare it to the maximum confidence we can obtain from a bigger graph.

1.1 Statistics of the hypothesis graph

Number of nodes and arcs of the hypothesis graph

```
In [6]: from hypotest.setup_hypothgraph import sample_graphs
```

```
INFO:rdflib:RDFLib Version: 4.2.1
```

```
In [7]: hypothgraph = sample_graphs.sample_hypothgraph()

In [8]: nb_nodes = len(hypothgraph.nodes())
        nb_arcs = len(hypothgraph.edges())

In [9]: print("number of nodes {}, number of arcs {}".format(nb_nodes, nb_arcs))
```

```
number of nodes 28, number of arcs 45
```

1.2 Hypothesis causal endpoints

Creation of a subgraph is a bit of *woodo* magic, we need to say what is the desired ratio of the paths from *source* to *target*, the causal endpoints of a hypothesis. So before we proceed we need the hypothesis configuration.

```
In [10]: import networkx as nx
```

```
In [11]: from hypotest.graph_generation import hypoth_conf
```

```
In [12]: source, target = hypoth_conf.generate_rich_endpoints(hypothgraph, min_nb_paths=6)
```

```
In [13]: print('number of paths from {} to {} is {}'.format(source, target, len(list(nx.all_simp
```

```
number of paths from http://plumdeq.xyz/ontologies/hypothesis/Loss_of_collagen to http://plumdeq
```

```
In [14]: conf = hypoth_conf.Hypoth_Conf(source, target, [])
```

1.3 Create a sub hypothesis graph

We deliberately choose a very small ratio of `on_boundary` paths, however we take some fair amount of paths within the hypothesis configuration.

```
In [15]: from hypotest.graph_generation import sub_hypothgraph
```

```
In [16]: ratio_on_boundary_paths = 0.01
         ratio_endpoints_paths = 0.1
```

```
In [17]: subgraph = sub_hypothgraph.generate_sub_hypothgraph(
             hypothgraph, source, target,
             ratio_endpoints_paths=ratio_endpoints_paths,
             ratio_on_boundary_paths=ratio_on_boundary_paths)
```

```
In [18]: nb_sub_nodes = len(subgraph.nodes())
         nb_sub_arcs = len(subgraph.edges())
```

```
In [19]: print("number of nodes {}, number of arcs {}".format(nb_sub_nodes, nb_sub_arcs))
```

```
number of nodes 19, number of arcs 26
```

1.4 Computing confidences

To compute the confidences in both sub and hypothgraphs we need to determine what are boundary interiors in both of the graphs.

1.4.1 Confidence computation

Confidence is computed as the sum of all weighted paths from `source` to `target` normalized to the number of these weighted paths. Suppose `source` is u and `target` is v.

$$Confidence(u,v) = \frac{\sum_{\pi_i(u,v)} weighted\_path(\pi_i)}{len(\pi(u,v))}$$

1.4.2 Boundary interiors of the two graphs

```
In [20]: from hypotest.graph_generation import boundary
```

```
In [21]: sub_boundary_interior = list(boundary.in_boundary_interior(subgraph, source, target))
```

```
In [22]: big_boundary_interior = list(boundary.in_boundary_interior(hypothgraph, source, target)
```

```
In [23]: print("Nb nodes in the boundary interior (subgraph) {}), nb nodes in full boundary inte
             len(sub_boundary_interior), len(big_boundary_interior)))
```

```
Nb nodes in the boundary interior (subgraph) 8), nb nodes in full boundary interior 9
```

```
In [24]: big_nb_paths = len(list(nx.all_simple_paths(hypothgraph, source, target)))
         small_nb_paths = len(list(nx.all_simple_paths(subgraph, source, target)))
         print("Nb of paths in big {} and in small {}".format(big_nb_paths, small_nb_paths))
```

```
Nb of paths in big 9 and in small 3
```

1.4.3 Confidence subgraph

```
In [25]: from hypotest.stats import confidences
```

Confidence in the hypothesis as we gradually add more evidenced nodes from the boundary, inside the subgraph only

```
In [26]: sub_confidences = confidences.confidence_spectrum(subgraph, source, target)
         sub_confidences
```

```
Out[26]: [0.0,
          0.6666666666666666,
          1.6666666666666667,
          2.6666666666666665,
          3.0,
          4.0,
          5.0,
          5.333333333333333,
          6.333333333333333]
```

```
In [27]: sub_confidences_normalized = confidences.confidence_spectrum(subgraph, source, target,
         sub_confidences_normalized
```

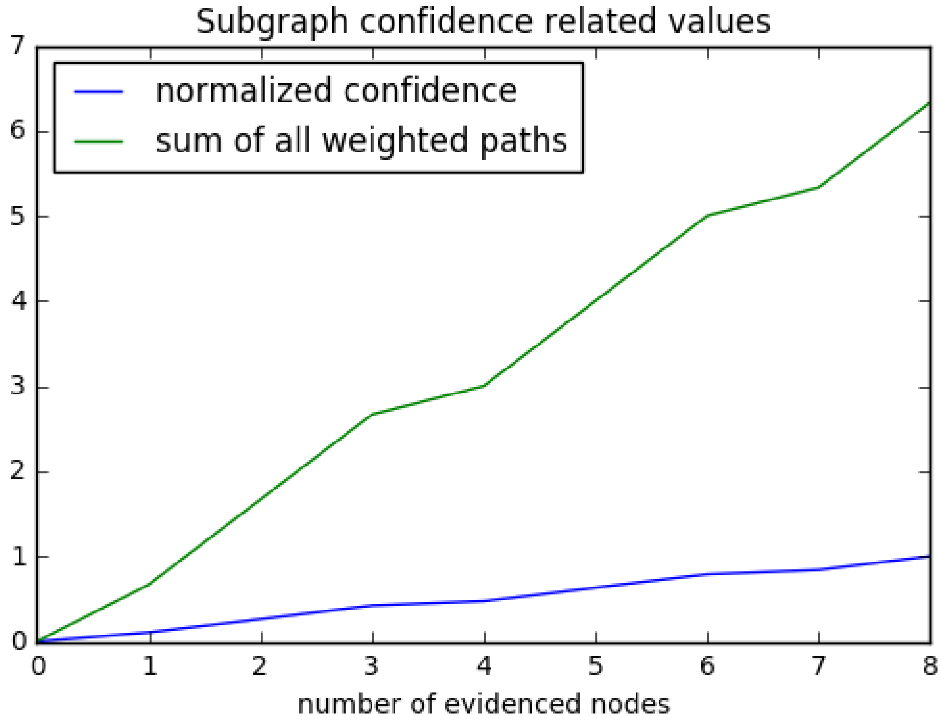
```
Out[27]: [0.0,
          0.10526315789473684,
          0.26315789473684215,
          0.42105263157894735,
          0.4736842105263158,
          0.6315789473684211,
          0.7894736842105263,
          0.8421052631578947,
          1.0]
```

1.4.4 Plot subgraph confidence related values

```
In [63]: import os
         path_to_figures = './images/subgraphs'
```

```
In [99]: subgraph_fig = plt.figure('1')
         plt.subplot('111')
         plt.plot(sub_confidences_normalized, label='normalized confidence')
         plt.plot(sub_confidences, label='sum of all weighted paths')
         plt.xlabel('number of evidenced nodes')
         plt.legend(loc='upper left')
         plt.title('Subgraph confidence related values')
```

```
Out[99]: <matplotlib.text.Text at 0x115ef9250>
```



Draw the figure with sub graph statistics

```
In [92]: # Make the first figure active
         subgraph_plot_png = os.path.abspath(os.path.join(path_to_figures, 'subgraph_plot.png'))
         subgraph_fig.savefig(subgraph_plot_png)
```

1.4.5 Confidence bigraph

```
In [29]: big_confidences = confidences.confidence_spectrum(hypothgraph, source, target)
         big_confidences
```

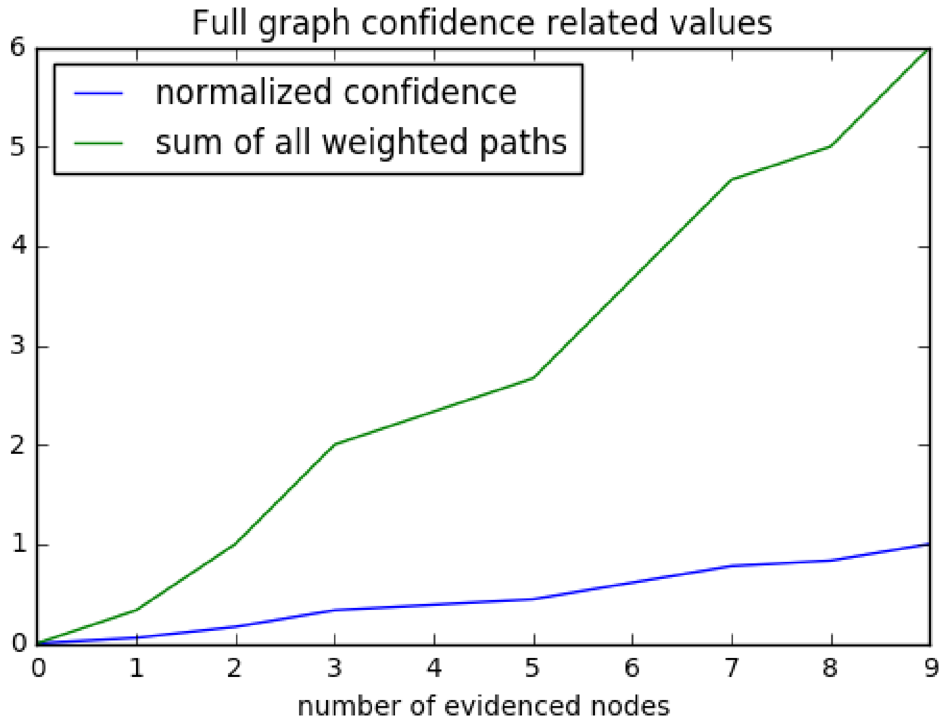
```
Out[29]: [0.0,
          0.3333333333333333,
          1.0,
          2.0,
          2.3333333333333335,
          2.6666666666666665,
          3.6666666666666665,
          4.666666666666667,
          5.0,
          6.0]
```

```
In [30]: big_confidences_normalized = confidences.confidence_spectrum(hypothgraph, source, targe
         big_confidences_normalized
```

```
Out[30]: [0.0,
          0.05555555555555555,
          0.16666666666666666,
          0.3333333333333333,
          0.3888888888888889,
          0.4444444444444444,
          0.611111111111111,
          0.7777777777777778,
          0.8333333333333334,
          1.0]
```

```
In [98]: biggraph_fig = plt.figure('2')
         plt.subplot('111')
         plt.plot(big_confidences_normalized, label='normalized confidence')
         plt.plot(big_confidences, label='sum of all weighted paths')
         plt.xlabel('number of evidenced nodes')
         plt.legend(loc='upper left')
         plt.title('Full graph confidence related values')
```

```
Out[98]: <matplotlib.text.Text at 0x115e1e650>
```



```
In [90]: # Make the first figure active
         biggraph_plot_png = os.path.abspath(os.path.join(path_to_figures, 'bigraph_plot.png'))
         biggraph_fig.savefig(biggraph_plot_png)
```

1.5 Relative confidence values

Here we check confidence values which can be obtained withing the small graph, normalized to the max confidence inside the small graph, as well as normalized to the max in the big graph

```
In [32]: import pandas as pd
```

```
In [33]: relative_spectrum = confidences.relative_confidence_spectrum(hypothgraph, subgraph, sou
```

```
In [34]: relative_spectrum_df = pd.DataFrame(relative_spectrum)
```

```
In [35]: relative_spectrum_df
```

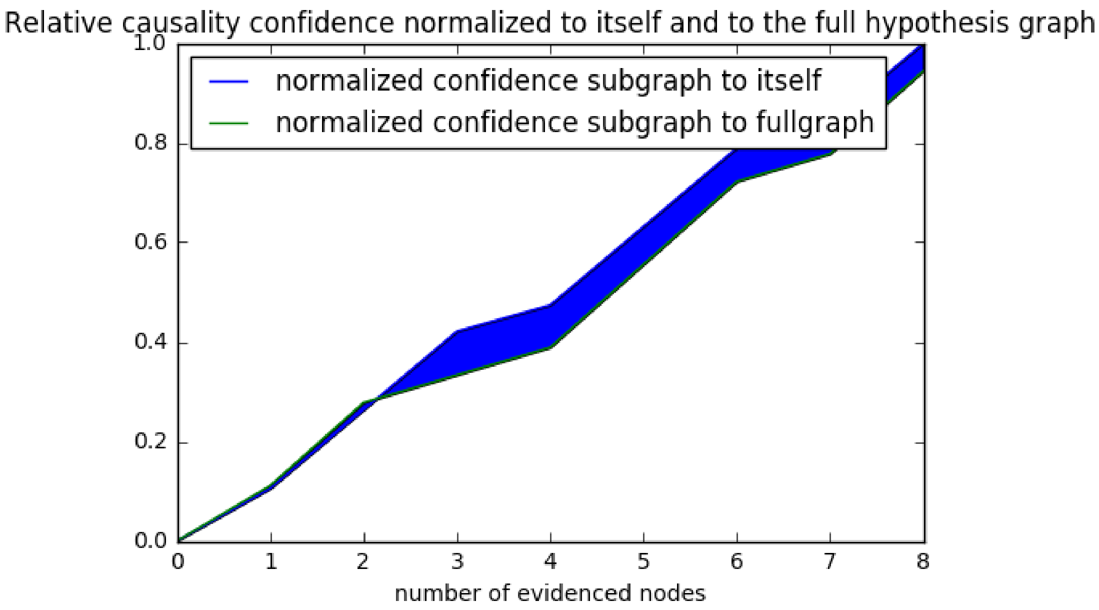
```
Out[35]:    big_confidence_normalized_spectrum  big_confidence_spectrum  \
         0                            0.000000                 0.000000
         1                            0.111111                 0.666667
         2                            0.277778                 1.666667
         3                            0.333333                 2.000000
         4                            0.388889                 2.333333
         5                            0.555556                 3.333333
```

```
6                          0.722222                 4.333333
7                          0.777778                 4.666667
8                          0.944444                 5.666667

   sub_confidence_normalized_spectrum  sub_confidence_spectrum
0                          0.000000                 0.000000
1                          0.105263                 0.666667
2                          0.263158                 1.666667
3                          0.421053                 2.666667
4                          0.473684                 3.000000
5                          0.631579                 4.000000
6                          0.789474                 5.000000
7                          0.842105                 5.333333
8                          1.000000                 6.333333
```

1.5.1 Plot relative confidence values

We plot relative normalized confidences

```
In [107]: relative_confidence_fig = plt.figure('3')
          plt.subplot('111')
          x = range(len(relative_spectrum_df['sub_confidence_normalized_spectrum']))
          plt.plot(relative_spectrum_df['sub_confidence_normalized_spectrum'],
                   label='normalized confidence subgraph to itself')
          plt.plot(relative_spectrum_df['big_confidence_normalized_spectrum'],
                   label='normalized confidence subgraph to fullgraph',
                   fillstyle='bottom')
          # show relative delta between the two
          plt.fill_between(x=x, y1=relative_spectrum_df['sub_confidence_normalized_spectrum'],
                           y2=relative_spectrum_df['big_confidence_normalized_spectrum'])
          plt.xlabel('number of evidenced nodes')
          plt.legend(loc='upper left')
          plt.title('Relative causality confidence normalized to itself and to the full hypothes

Out[107]: <matplotlib.text.Text at 0x11660d150>
```



```
In [95]: # Make the first figure active
         relative_confidence_plot_png = os.path.abspath(os.path.join(path_to_figures, 'relative_
         relative_confidence_fig.savefig(relative_confidence_plot_png)
```

1.6 Draw this configuration

```
In [37]: from hypotest.io import write_dot
```

```
In [64]: from IPython.display import Image
```

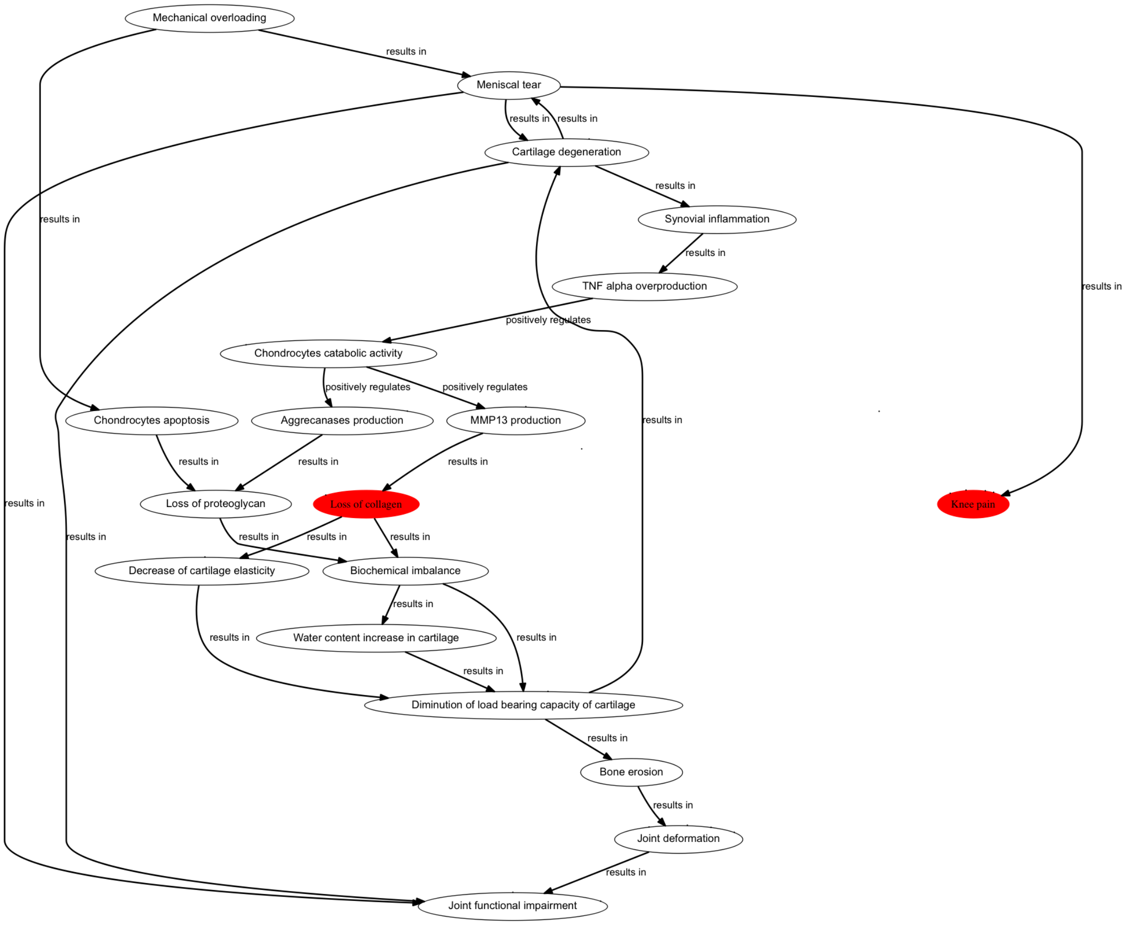
1.6.1 Draw subgraph, by including positions of the nodes of the big graph

```
In [42]: small_dot = os.path.join(path_to_figures, 'small.dot')
         small_png = os.path.join(path_to_figures, 'small.png')

         with open(small_dot, 'w') as f:
             write_dot.hypothgraph_with_invisible_to_dot(hypothgraph, subgraph, conf, stream=f)
```

```
In [43]: !dot -Tpng -o $small_png $small_dot
```

```
In [46]: Image(small_png)
```

```
Out[46]:
```



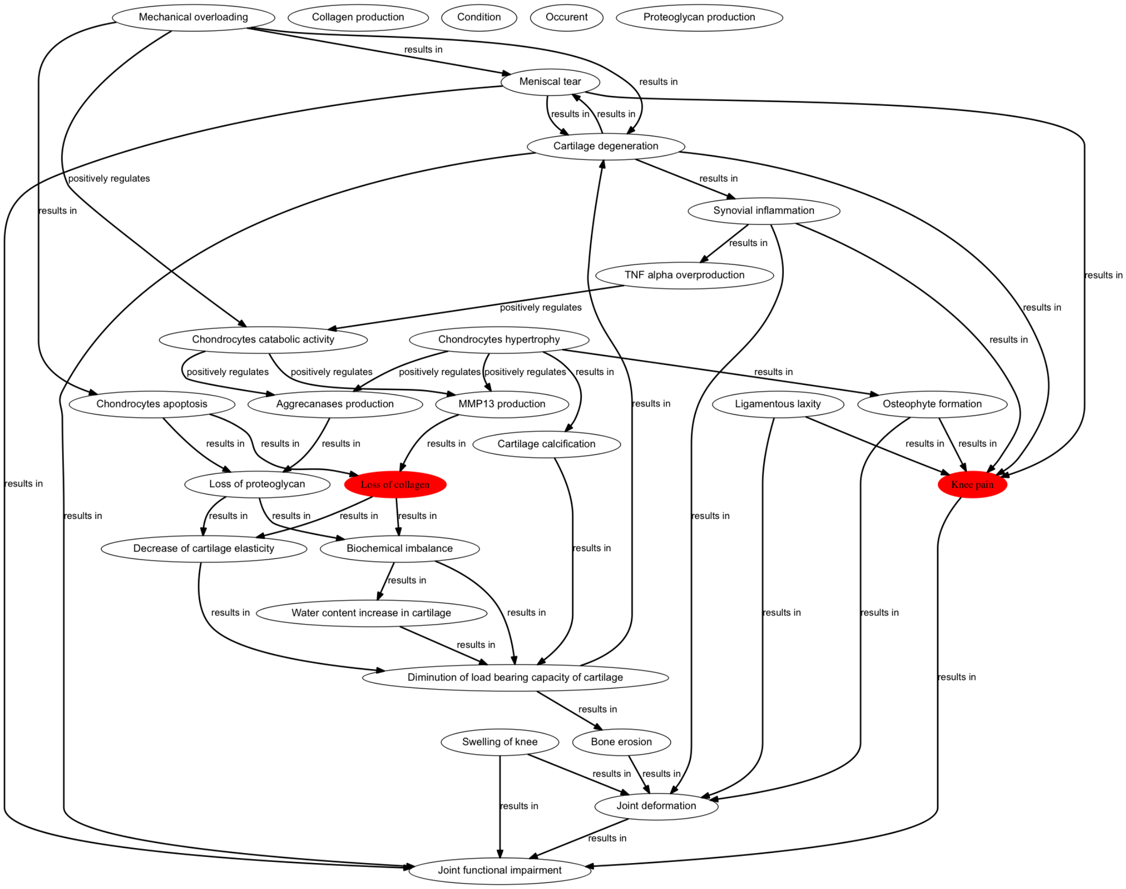
1.6.2 Draw big graph

```
In [51]: big_dot = os.path.join(path_to_figures, 'big.dot')
         big_png = os.path.join(path_to_figures, 'big.png')

         with open(big_dot, 'w') as f:
             write_dot.hypothgraph_to_dot(hypothgraph, conf, stream=f)
```

```
In [52]: !dot -Tpng -o $big_png $big_dot
```

```
In [53]: Image(big_png)
```

```
Out[53]:
```



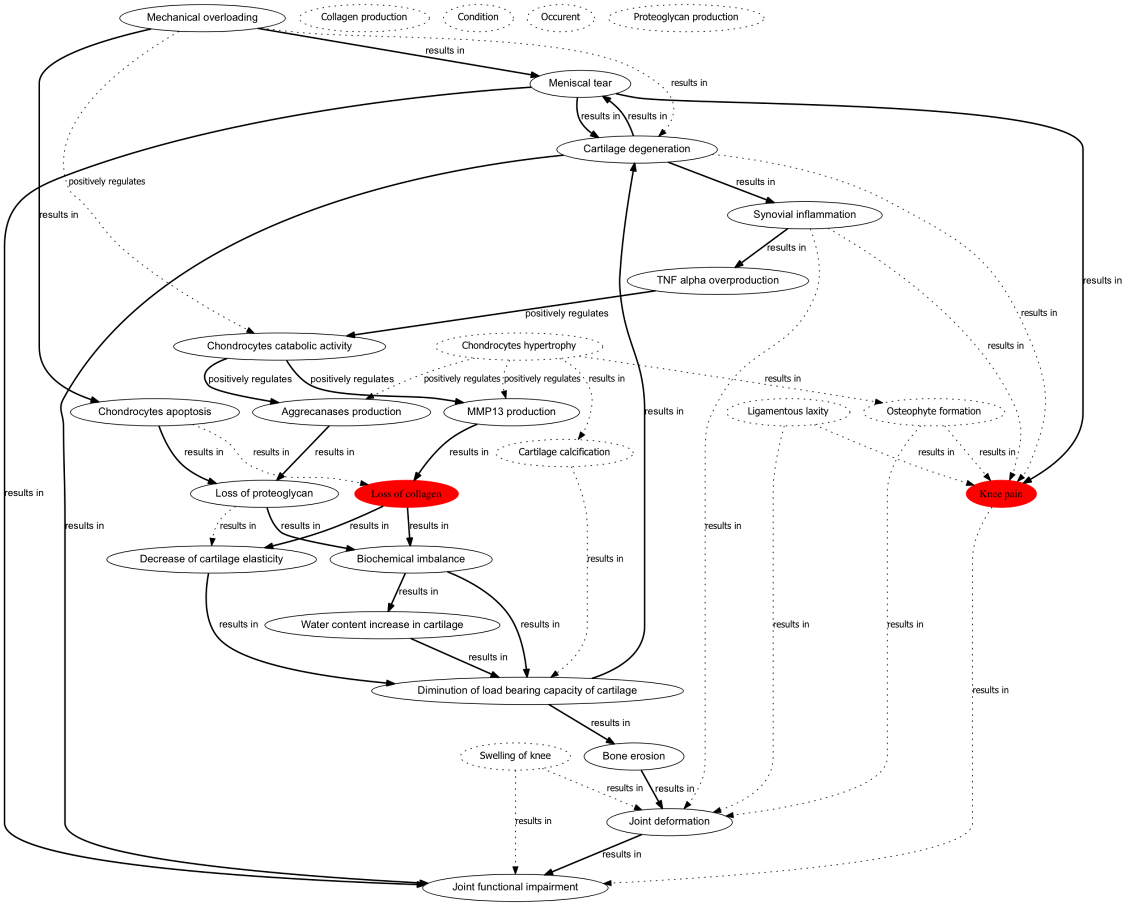
1.6.3 Draw superimposed

```
In [54]: superimposed_dot = os.path.join(path_to_figures, 'superimposed.dot')
         superimposed_png = os.path.join(path_to_figures, 'superimposed.png')

         with open(superimposed_dot, 'w') as f:
             write_dot.big_small_to_dot(hypothgraph, subgraph, conf, stream=f)
```

```
In [55]: !dot -Tpng -o $superimposed_png $superimposed_dot
```

```
In [56]: Image(superimposed_png)
```

```
Out[56]:
```


Mechanical overloading
Collagen production
Condition
Occurent
Proteoglycan production
Meniscal tear
Cartilage degeneration
Synovial inflammation
TNF alpha overproduction
Chondrocytes catabolic activity
Chondrocytes hypertrophy
Chondrocytes apoptosis
Aggrecanases production
MMP13 production
Ligamentous laxity
Osteophyte formation
Cartilage calcification
Loss of proteoglycan
Loss of collagen
Knee pain
Decrease of cartilage elasticity
Biochemical imbalance
Water content increase in cartilage
Diminution of load bearing capacity of cartilage
Swelling of knee
Bone erosion
Joint deformation
Joint functional impairment
results in
positively regulates