ID2222 Assignment report 3

Authors:

- Marco Dallagiacoma marcoda@kth.se
- Roberto Bampi bampi@kth.se

The purpose of this first assignment is to implement a hyperloglog sketch and use it to implement an algorithm to efficiently compute graph centrality metrics using a small amount of memory.

The code is implemented as a pure-python module and it is compatible with python versions 3.3 and above. To run it, the use of the `pypy` interpreter is advised as the just-in-time compiler allows the code to achieve better performance and to complete earlier. The easiest way to run it is to create a virtualenv first and then install the package into it.

```bash
# assuming the current directory contains the setup.py file
~ $ virtualenv -p $(which python3) .venv
~ $ source .venv/bin/activate
~ (venv) $ pip install -e .
```

Once the steps above are complete, the package is now installed and ready to run. The program can be run by invoking the `run-hyperball` command which takes only one positional parameter: the path to a file containing a graph. The graph must be composed of multiple lines, each containing (at least) two columns separated by spaces. Each colum represent one edge of the graph.

Solution

The program is implemented as a Python module and divided into two files. The algorithms are implemented in `hyperloglog.py` and `hyperball.py` respectively.

Sample run

Below is the output of running `run-hyperball` on the moreno-oz network. Nodes in the network represent people and edges friendships. The output is as follows:

```Bash
~ $ run-hyperball moreno_oz/out.moreno_oz_oz
node    closeness  lin        harmonic
  54    0.00352    205.427    124.327
 169    0.00344    200.908    171.180
  71    0.00317    185.085    132.712
 126    0.00305    178.262    112.034
 115    0.00305    177.851    150.300
 185    0.00304    177.707    102.167
  50    0.00300    174.921    123.551
 138    0.00300    174.886    137.556
  73    0.00295    172.147    152.589
 151    0.00294    171.478    152.584
  51    0.00288    168.379    131.108
 133    0.00288    167.831    140.503
 125    0.00285    166.307    138.909
  85    0.00284    165.755    137.721
 156    0.00281    164.300    126.630
... more output ...
```

Algorithms

Hyperloglog

Hyperloglog is a algorithm designed to solve the problem of estimating the cardinality of a set S when it is impractical to store all of the elements. The core assumption is that a hyperloglog sacrifices the possibility of checking for the presence of a single item (`contains`) to estimate the cardinality of very large sets, with a configurable error rate, using very little memory.

In order to estimate the cardinality hyperloglog stores a number of buckets containing the number of leading zeroes in the binary string that is the hash of the original elements. Given that, in the a hash of a value (produced with a good hash function) each bit can be either 0 or 1, the probability of a sequence of zeroes of a given length n is $(\frac{1}{2})^n$. This probability gives a probabilistic size of S but is very subject to outliars. To counteract this, multiple buckets are used and the likely size is computed by averaging all the value using a harmonic mean.

Hyperball

Hyperball is an algorithm that estimates various centrality measures such as *closeness centrality* and *lin distance* using little memory by leveraging hyperloglog counters.

By associating every node with a counter and repeatedly merging the counter with that of its neighbors, the algorithm is able to converge and compute all of the metrics necessary to build the centrality measurements.

Questions

What were the challenges you have faced when implementing the algorithm?

As with any probabilistic algorithm it is very challenging to test the results and bugs are often very hard to track down. Moreover, the use of Python as the implementation language, while convenient, results in a very slow algorithm in practice which further slows the feedback loop.

Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.

Yes, in the main loop, the new hyperloglog counter can easily be calculated in parallel for each node in the graph, massively speeding up the computation. This is possible because the merge of the new hyperloglog counters happens as an atomic operation at the end of the loop for every generation `t`.

Does the algorithm work for unbounded graph streams? Explain.

Given that the algorithm uses a small but fixed amount of memory for every node in the graph, the memory would grow forever if the graph is unbounded. Addition of edges should not, however, cause problems for any practical amount of edges due to the use of a 64 bit hash funciton and 6 bit registers in the hyperloglog counters.

Does the algorithm support edge deletions? If not, what modification would it need? Explain.

Given that the measures are computed as hyperloglog counters for each node, and hyperloglog sacrifice the ability of accessing single items, the delete operation is not supported.