Carnegie Mellon University
15-826 Multimedia Databases & Data Mining
Fall 2014 - C. Faloutsos

Default-project Description: Graph Mining using SQL

Contact TA: Neil Shah , neilshah@cs.cmu.edu

1 Introduction - Problem description

Do we need an additional language, to do graph manipulations? The provided '*graphMiner*' package shows that SQL is enough to answer all the major graph-mining questions. The brief package description is as follows:

- GIVEN a graph of (source, destination) pairs on the disk
- WRITE code in SQL
- TO IMPLEMENT basic matrix and vector operations (L2 norm, dot product, matrix multiplication)

These form the basis used to build more advanced graph mining techniques including operations that we will cover in class, or are described in the papers in Section 3.

- Degree distribution
- PageRank
- Weakly connected components
- Eigenvalue computation (via Lanczos-SO and QR algorithms)
- "Fast" belief propagation
- Triangle count

In this project, you will extend this work to incorporate unit-tests, the so-called k-cores graph-mining algorithm and improved SQL indexing methods. Finally, you will apply all the algorithms (including those listed above) on many real graphs and summarize the results ('*which patterns are followed by most graphs*?', '*which graphs/nodes seem like anomalies*?', etc.)

Notes:

- Assume that the input file is in edge-list form `(source-id, destination-id)`.
- Use *postgres*, since it has the best query optimization among the open-source RDBMSs, and since *graphMiner* uses postgres anyway.

2 Data

There are some terrific repositories, with several overlapping datasets:

- http://konect.uni-koblenz.de/networks/ from the KONECT project.
- http://snap.stanford.edu/data/index.html from the SNAP project.
- http://www-personal.umich.edu/~mejn/netdata/ from Prof. Mark Newman.

Start with a few, small, datasets for phase 1, and we will create local mirrors here, to save you downloading effort and bandwidth. Some of the datasets are large (Gbs) - please avoid them, until we set up the local mirror.

3 Introductory Papers

- The PEGASUS paper with GIM-V http://www.cs.cmu.edu/~ukang/papers/PegasusKAIS.pdf Discusses PageRank and connected components.
- the follow-up paper of GBASE http://www.cs.cmu.edu/~ukang/papers/GbaseVLDBJ2012.pdf
- 'HADI' [TKDD'11] http://web.kaist.ac.kr/~ukang/papers/HadiTKDD2011.pdf for diameter and radius estimation,
- 'HEIGEN' [PAKDD'11] for eigenvalue computation,
- For Fast Belief Propagation, check [PKDD'11] http://www.cs.cmu.edu/~dkoutra/papers/fabp_pkdd2011.pdf
- For the k-cores algorithm and applications, check http://arxiv.org/pdf/cs/0504107v2
- Also skim the rest of the papers on the Pegasus project web site http://www.cs.cmu.edu/~pegasus/publications.htm

4 Tasks and Deliverables

Here is the detailed list of deliverables and point distribution. The maximum grade in each phase is 100, and the weights of each phase are as announced (10%, 10% , 80%)

Phase 1: 10% of project weight, max score: 100

Your write-up should be about 6-8 pages.

1. (**30 pts**) Complete a literature survey: 3 paper reviews per team member, from the introductory papers above (Section 3). Paper reviews should consist of
   (a) a *summary* of the main idea of the paper (in your *own* words - cutting-and-pasting text from the paper is **plagiarism**)
   (b) list of *shortcomings*, that you think that future research could address.
2. (**35 pts**) Implement the k-cores algorithm – specifically, compute the $k = 5$ cores and return the connected components. Your output should simply be a list of nodes in the $k = 5$ decomposition in addition to which connected component each node falls in. Do not worry about numbering schemes for the connected components – just seek to ensure that nodes from the same connected component are identified with the same connected component ID.

3. (**25 pts**) Write (at least) 5 unit tests to ascertain functionality for the $k = 5$ cores algorithm you wrote, degree distribution (in/out) and connected components. The unit tests would be, say: 5-node-clique graph as the input, 10-node chain, etc. Just use simple graphs for which it is easy to infer and check properties.
4. (**10 pts**) Run the *graphMiner*[1] code including your algorithms on 2 datasets ($cit - Patents$ and $soc - Epinions1$ from SNAP) and report the results. Please use the set-up code at `http://www.cs.cmu.edu/~christos/courses/826.F14/project-default-graphs/826-proj-setup.tar.gz` which will setup the necessary components to run *graphMiner* on the GHC machines and run an included demo. **Read the README – it will explain the specifics on how to run the code and manage postgres for future sessions.** *IMPORTANT:* we will test and grade your code on the GHC machines. Other machines may not have necessary dependencies installed (probably `flex` and `bison`)

Phase 2: 10% of project weight, max score: 100

Your write-up should be about 10-15 pages (including your Phase 1 write-up)

1. (**60 pts**) Experiment with various indexing options (`create index`) in postgres to find the fastest setting for graph mining algorithms. Describe the settings you tried, the one you think is best, and the wall-clock times that support your opinion.
2. (**40 pts**) Run the *graphMiner* code including your algorithms and indexing procedures of choice on the 10 biggest graphs and report the results. Give the plots (degree distributions, scatter-plots[2], etc) and a list of your observations.

Phase 3: 80% of project weight, max score: 100

Your write-up should be about 20-30 pages long (including all previous write-ups).

1. (**10 pts**) Provide user-manual documentation about the k-cores code you implemented – check the report from the previous semester for a documentation example. You can find the report in the doc directory in the *graphMiner* archive.
2. (**15 pts**) Packaging of code and documentation: Provide a tar-file with your code, the unit-tests, and your report (latex sources). Typing `make`
   - should generate all the plots in your report,
   - it should run all your unit-tests,
   - and it should create the pdf of your report
3. (**75 pts**) Report results on *20* graphs, in summarized form, that is, report global patterns, and strangely-behaving datasets for the various metrics – for example, all datasets might have radius x except a select few, or all graphs might follow a power-law with a given exponent range except a select few). Describe how you did the summarization (it is non-trivial!) as

[1] Courtesy of Nijith Jacob and Sharif Doghmi, who took this class in a previous year – `http://www.cs.cmu.edu/~christos/courses/826.F14/project-default-graphs/graphminer.tar.gz`

[2] You could use heatmap producing code from Danai Koutra - `http://www.cs.cmu.edu/~dkoutra/CODE/heatmap.tar` - or another package if you prefer

well as the anomaly detection (also non-trivial!) 3 points are given for each dataset studied and 1 point for every observation made.

4.1 Details on deliverables:

For every phase, please:

- Hand-in a hard copy of your write-up, typed, 12pt font, neat and with pictures if applicable
- and a tar-file with your code, including a `makefile`)

More details:

- Use the LaTeX template – `http://www.cs.cmu.edu/~christos/courses/826-resources/PROJECT-SAMPLES/samplePaper.tar.gz` Adapt the section headers, accordingly, eg.,
  - introduction
  - ph1: paper-reviews by person1
  - ph1: paper-reviews by person-2
  - ph1: description of unit tests
  - ph2: description of algorithms
  - ph2: indexing settings and results
  - ph3: summary of patterns on 100 graphs
  - ph3: exceptions and anomalies
  - etc
- Provide a plan of activities and time estimates per group member.
- List which group member did (or will do) what.
- Check grammar and syntax (small penalty for each typo/grammar error).
- Keep the graded reports and attach them, every time. That is for Phase 2, attach the graded Phase 1 report; for Phase 3, attach all previous, graded, reports.

4.2 Important: code packaging and ease-of-use

Being a mainly implementation project, your code should be

- neatly packaged, ie., no extraneous files (*.o, *.pyc, *.aux, *.log)
- able to run on the linux-andrew machines,
- with unit-tests,
- with a `makefile` - typing `make` should run a small demo
- with a (brief) user's manual
- and with an easy interface: e.g., from the Unix prompt
  `graphmine my-edge-list.csv`
  with appropriate arguments should load the given input file into postgres, run all your queries, and generate all the plots for above tasks.

4.3 Logistics - reminders

- *Academic Attribution*: Whenever you use ideas, text, code, algorithms, from someone else, *please cite this person, paper, or url*. Copying without attribution constitutes *plagiarism* leading to severe penalties (failing the class, expulsion, etc).
- All projects will be done in **groups of two** (with exceptions only under special circumstances, after instructor's permission.)