Carnegie Mellon University
15-826 Multimedia Databases & Data Mining
Spring 2016 - C. Faloutsos

Default-project Description: Graph Mining using SQL

Designed by : Di Jin and Varshaa Naganathan - updated 2/29 and 4/16

1 Introduction - Problem description

Do we need an additional language, to do graph manipulations? The provided '`graphMiner`' package[1] shows that SQL is enough to answer all the major graph-mining questions:

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

In this project, you and your team mate are to extend this work in 4 directions:

1. you will do unit-tests, for the provided operations
2. you will implement the so-called k-core graph-mining algorithm
3. you will experiment with improved SQL indexing methods and
4. you will apply all the algorithms (including those listed above) on many real graphs and summarize the results ('*what knowledge can you discover in a certain graph*?', '*which patterns are followed by most graphs*?', '*which graphs/nodes seem like anomalies*?', etc.)

Notes:

- *Team size:* Two members per team.
- Assume that the input file is in edge-list form `(source-id, destination-id)`.
- **Un-directed:** whenever necessary, make sure your graph is un-directed, by mirroring the appropriate edges - `graphMiner` has a command-line switch for that
- Use *postgres*, since it has the best query optimization among the open-source RDBMSs, and since `graphMiner` uses postgres anyway.

[1] Courtesy of Nijith Jacob and Sharif Doghmi, who took this class in an earlier year – `http://www.cs.cmu.edu/~christos/courses/826.S16/project-default-graphs/graphminer.tar.gz` For your convenience, use the set-up code at `http://www.cs.cmu.edu/~christos/courses/826.S16/project-default-graphs/826-proj-setup.tar.gz`

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.

Test your implementation with some of these datasets for phase 1, but using synthetic datasets would also be fine (refer to **Phase 1** for more details).

3 Introductory Papers and definitions

Next we give two lists of papers. You should read the first, but you should choose papers to survey, from the second one.

3.1 `graphMiner` papers

Here is the list of papers that `graphMiner` is based on - you should at least skim them, to understand the package. For your convenience, all urls are *click-able*.

- The PEGASUS paper with GIM-V: It discusses PageRank and connected components.
  `http://www.cs.cmu.edu/~ukang/papers/PegasusKAIS.pdf`
- the follow-up paper of GBASE:
  `http://www.cs.cmu.edu/~ukang/papers/GbaseVLDBJ2012.pdf`
- 'HADI' [TKDD'11] for diameter and radius estimation,
  `http://web.kaist.ac.kr/~ukang/papers/HadiTKDD2011.pdf`
- 'HEIGEN' [PAKDD'11] for eigenvalue computation:
  `http://www.cs.cmu.edu/~ukang/papers/HeigenPAKDD2011.pdf`
- For Fast Belief Propagation, check [PKDD'11]:
  `http://web.eecs.umich.edu/~dkoutra/papers/fabp_pkdd2011.pdf`
- Skim the rest of the papers on the Pegasus project web site:
  `http://www.cs.cmu.edu/~pegasus/publications.htm`

3.2 Papers for your survey

Here is the list of papers you should choose from, in your survey. You may survey additional papers, but check with the instructor first.

- Belief Propagation
  - Generalization of fast-belief-propagation [VLDB'15]
    `http://arxiv.org/pdf/1406.7288`
- k-cores and generalizations:
  - k-cores algorithm and applications:
    `http://arxiv.org/pdf/cs/0504107v2`

  - a whole web site on k-cores and generalizations
    http://www.graphdegeneracy.org/
- The Vertexica paper, also arguing that SQL is good for graph mining
  http://www.vldb.org/pvldb/vol7/p1669-jindal.pdf
- And some graph visualization and graph summarization papers
  - Visualization
    http://hcil2.cs.umd.edu/trs/2012-29/2012-29.pdf
  - static graph summarization: [SDM'14]
    http://web.eecs.umich.edu/~dkoutra/papers/VoG_journal.pdf
  - time-evolving graph summarization [KDD'15]
    http://web.eecs.umich.edu/~dkoutra/papers/Timecrunch_KDD15.pdf

3.3 Definitions of k-core concepts

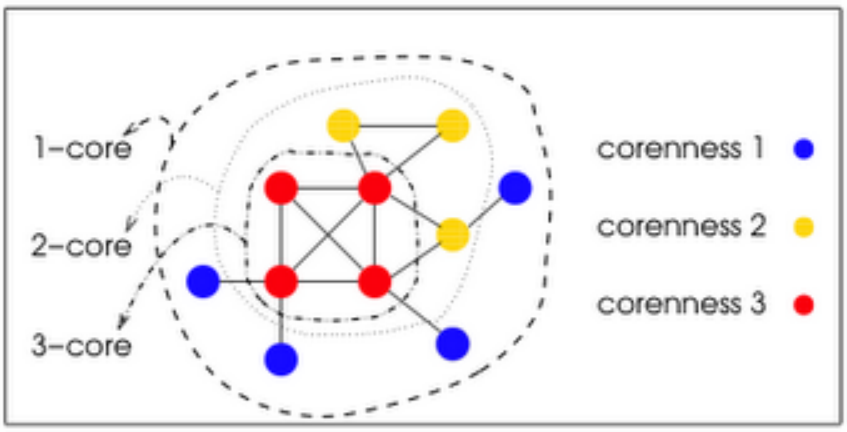

Figure 1: Example of k-core decomposition, coreness, and degeneracy. [From arxiv:cs/0504107v2, Alvarez-Hamelin et. al., 2005]

The definitions of k-core concepts are described in the arxiv paper above http://arxiv.org/pdf/cs/0504107v2.pdf. For your convenience, we repeat Figure 1 from there, in our Figure 1. The blue nodes have *coreness* =1, and so on; the whole graph has *degeneracy* = 3, since that is the highest coreness among its nodes (thanks to the red nodes). Formally, we have:

Definition 1 (Coreness of a node) *A node n in an undirected graph has coreness k, if it has k or more neighbors that have coreness k or higher, and k is the maximum such integer for node n.*

Definition 2 (k-core of a graph) *For a given undirected graph, the k-core is the induced subgraph that contains all the nodes with coreness $\geq k$, (and the edges between them).*

Definition 3 (Degeneracy D of a graph) *This is the highest coreness among the nodes of the graph.*

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: at least 3 paper reviews per team member, from the introductory papers above (Section 3). Paper reviews should consist of
   (a) the *problem definition* that the paper is addressing
   (b) a *summary* of the main idea of the paper (in your *own* words - cutting-and-pasting text from the paper or any other source, is **plagiarism**)
   (c) list of *shortcomings*, that you think that future research could address.
2. (**35 pts**) Implement the k-degeneracy algorithm (aka, k-core algorithm)[2] using the so-called "shaving" method, that is, repeatedly delete nodes with degree $< k$, until nothing changes. Specifically, the code should:
   - compute the coreness k of each node, and store them in a table;
   - compute and return the degeneracy D of the input graph.
3. (**25 pts**) Write (at least) 5 unit tests for (a) the k-core algorithm you wrote, (b) for the degree distribution (in/out), and (c) for the connected components. The unit tests could be, say: a 5-node-clique graph as the input, a 10-node chain, etc. Use simple graphs for which it is easy to infer and check properties. In your report, include the adjacency matrices of the unit test graphs and the outputs of your k-core algorithm.
4. (**10 pts**) Run the `graphMiner` code, including your algorithms, on 2 datasets
   - `soc-Slashdot0811` and
   - `soc-Epinions1`

   from SNAP, and report the results. Specifically, report the following 3 results:
   (a) the core values of the following 5 testing nodes with ID:

   $$0$$
   $$17$$
   $$9422$$
   $$18475$$
   $$27763$$

   (b) the degeneracy value of the input graph, and
   (c) the core value distribution (i.e., the PDF: coreness vs. count).

[2]Definition of k-degeneracy: `https://en.wikipedia.org/wiki/Degeneracy_(graph_theory)`

Important - for easy set-up: Please use the set-up code mentioned in the introduction, and repeated in this footnote here[3]. This will setup the necessary components to run `graphMiner` on the GHC machines and it will also run an included demo.

- *Read* the `README` file in the tar-file – it will explain the specifics on how to run the code and manage postgres for future sessions.
- *Grading:* we will test and grade your code on the GHC machines. Use the GHC machine (`ghcXX.ghc.andrew.cmu.edu`) and postgres port ID assigned to you, on 'blackboard'
- In the remote case that your assigned `ghc` machine creates problems, try another one in the range `ghc25 - ghc86`, but please use your assigned port number.
- *Specify* the right postgres port in the `setup-proj.bash` file to avoid conflicts with another student's postgres installation (see `README`).

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. Possible choices are: (a) a clustering index on `source` (b) a non-clustering index on `source`, (c) a composite index on `source, destination`. You may also consider possible orderings of the nodes (eg., by pagerank, or coreness, or by the 'slash-N-burn' method [Kang+ ICDM'11] `http://www.cs.cmu.edu/~ukang/papers/sb_icdm2011.pdf` or some combination of the above. Ideally, whatever you do, should be done through SQL. The conjecture is that if we order the edges carefully, all/most of the graph mining algorithms will be faster.

   (New: added 2/29/2016) - One-person teams: *Such teams need only do (a),(b) and (c), for two cases: (A) nodes in random order and (B) nodes in pagerank order. That is, such teams should give results for 5 graphs, for all 6 cases above ($\{$ a, b, c$\} \times \{$ A, B $\}$).*
2. (**40 pts**) Run the `graphMiner` code including your algorithms and indexing procedures of choice on 5 graphs (you may have to do preprocessing) and report the results. Give the plots (degree distributions, scatter-plots[4], 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-degeneracy 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.

[3] `http://www.cs.cmu.edu/~christos/courses/826.S16/project-default-graphs/826-proj-setup.tar.gz`

[4] You may use heatmap-producing code from Danai Koutra - `http://www.cs.cmu.edu/~christos/SRC/heatplot.tar`

2. (**15 pts**) Packaging of code and documentation: Provide a tar-file with your code, the unit-tests, and your report (latex sources). Make sure it matches the check-list in subsection 4.2.
3. (**75 pts**) Report results on *15* graphs (you may have to do preprocessing), 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 well as the anomaly detection (also non-trivial!) 4 points are given for each dataset studied and 1 point for every observation made.

(New: added 2/29/2016) - One-person teams: *Such teams have to do all these steps, but only for 5 graphs, instead of 15.*

Phase 3: More details

(New: added 4/16/2016)

- **Plots**: for each graph, we expect 8 plots for 0.5 points per plot - we recommend the first eight below, but you can replace some of them with other choices (see graph mining textbook for ideas, chapters 3,4,5 - give the reason for your choice of plot). The plots come in two flavors: (a) distribution plots (PDF or CCDF) and (b) scatter-plots.
  1. degree distribution (S-1 pattern in textbook)
  2. pageRank distribution,
  3. eigen/singular value distribution (see textbook, Figure 3.3)
  4. triangle distribution (see textbook, Figure 3.5(a))
  5. connected-component-size distribution (see p. 32 of graph mining lecture foils.
  6. core-ness value distribution
  7. degree vs pageRank scatterplot
  8. degree vs count-of-triangles scatterplot (see textbook, Figure 3.5(b))

  You may replace some of the above with
  - scatter-plot of in-degree vs authority score (HITS) if the graph is directed; ditto for out-degree vs hubness score. (see Figure 7, in [Jiang+, KDD'14])
  - eigenspokes plots (see page 16 of graph mining lecture foils).
  - degree vs weight scatterplot (W-1 pattern in textbook - if a graph is weighted)
  - radius plot
  - or some other plot that you may deem promising
- **Summarization:** you could/should combine the distribution plots, for your 15 graphs.
- **Observations:** we expect 15 observations, total. Some graphs may lead to no observation; some others may lead to several observations.
- *Suggestion - Outlier detection:* You may do that, by visual inspection. *No need* to use automatic outlier detection methods (like LOF, GFADD, etc).
- *Suggestion - Log scales:* For most plots, it would be best to use logarithmic scales - unless you give a good reason to do otherwise.
- *Hint - Heatmaps:* - if there are too many points on your plot, and/or if there is severe over-plotting, please use heatmaps - eg., try the script by Prof. Koutra and the instructor.

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 at:
  `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 15 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 nicely packaged - follow the template at

`http://www.cs.cmu.edu/~christos/courses/826-resources/PROJECT-SAMPLES/samplePackage.tar.gz`

That is, you are expected to deposit a `tar`-file, matching the following check-list:

1. neatly packaged, ie., no extraneous files (*.o, *.pyc, *.aux, *.log)
2. able to run on the linux-andrew machines,
3. with unit-tests,
4. *with out* huge data files (instead, use, e.g., `wget` in your `makefile`, to pull all such files, dynamically).
5. with a `makefile`
   - typing `make` should run a small demo;
   - `make paper.pdf` should create your report
   - `make clean` should eliminate all the derived files (*.o, *.class, *.aux, etc)
   - `make all.tar` should create a tar-file, ready for distribution
6. with brief usage instructions, in the `README` file

7. with your report in the directory `./DOC`
8. and the code should have 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 / Plagiarism*: 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).
- *Team size*: As mentioned, all projects will be done in **groups of two** (with exceptions only under special circumstances, after instructor's permission.)
- *Poster: No need* to do a poster, in contrast to the non-default projects. If you really want to do a poster, you are welcome to do so - please notify the instructor, to plan for easels, thumb-tacks etc.