CHAPTER 17

Case Studies

Graph Analysis Chimpanzee Style

Using Pig and Wukong to Explore Billion-edge Network Graphs

Basics

Networks at massive scale are fascinating. The number of things they model are extremely general: if you have a collection of things (that we'll call nodes), they are related (edges), and the nodes and edges tell a story (node / edge metadata), you have a network graph.

I started the Infochimps project, a site to find, share or sell ay dataset in the world. At Infochimps, we've got a whole bag of tricks ready to apply to any interesting network graph that comes into the collection. We chiefly use Pig (described in chapter XXXX) and Wukong (*http://github.com/mrflip/wukong*), a toolkit we've developed for Hadoop streaming in the Ruby programming language. They let us write simple scripts like the ones below—almost all of which fit on a single printed page—to process terabyte-scale graphs. Here are a few datasets that come up in a search for "network" on infochimps.org*:

- A social network, such as Twitter or Facebook. We somewhat impersonally model people as nodes, and relationships (`@mrflip` is friends with `@tom_e_white`) or actions (`@infochimps` mentioned `@hadoop`) as edges. The number of messages a user has sent and the bag of words from all those messages are each important pieces of node metadata.
- A linked document collection such as Wikipedia or the entire web†. Each page is a node (carrying its title, view count and categories as node metadata). Each hy-

* *http://infochimps.org/search?query=network*

† *http://www.datawrangling.com/wikipedia-page-traffic-statistics-dataset*

perlink is an edge, and the frequency at which people click from one page to the next is edge metadata.

- The connections of neurons (nodes) and synapses (edges) in the *C. elegans* roundworm.‡
- A highway map, with exits as nodes, and highway segments as edges. The Open Street Map project's dataset has global coverage of place names (node metadata), street number ranges (edge metadata), and more.§
- Or the many esoteric graphs that fall out if you take an interesting system and shake it just right. Stream through a few million Twitter messages, and emit an edge for every pair of non-keyboard characters occurring within the same message. Simply by observing "often, when humans use 最, they also use 近" you can recreate a map of human languages (see Figure 17-1).

What's amazing about these organic network graphs is that given enough data, a collection of powerful tools are able to *generically* use this network structure to expose insight. For example, we've used variants of the same algorithm‖ to do each of:

- Rank the most important pages in the Wikipedia linked-document collection. Google uses a vastly more refined version this approach to identify top search hits.
- Identify celebrities and experts in the Twitter social graph. Users who have many more followers than their "trstrank" would imply are often spammers.
- Predict a school's impact on student education, using millions of anonymized exam scores gathered over five years.

Measuring Community

The most interesting network in the Infochimps collection is a massive crawl of the Twitter social graph. With more than 90 million nodes, 2 billion edges, it is a marvelous instrument for understanding what people talk about and how they relate to each other. Here is an exploration, using the subgraph of "People who talk about Infochimps or Hadoop"#, of three ways to characterize a user's community:

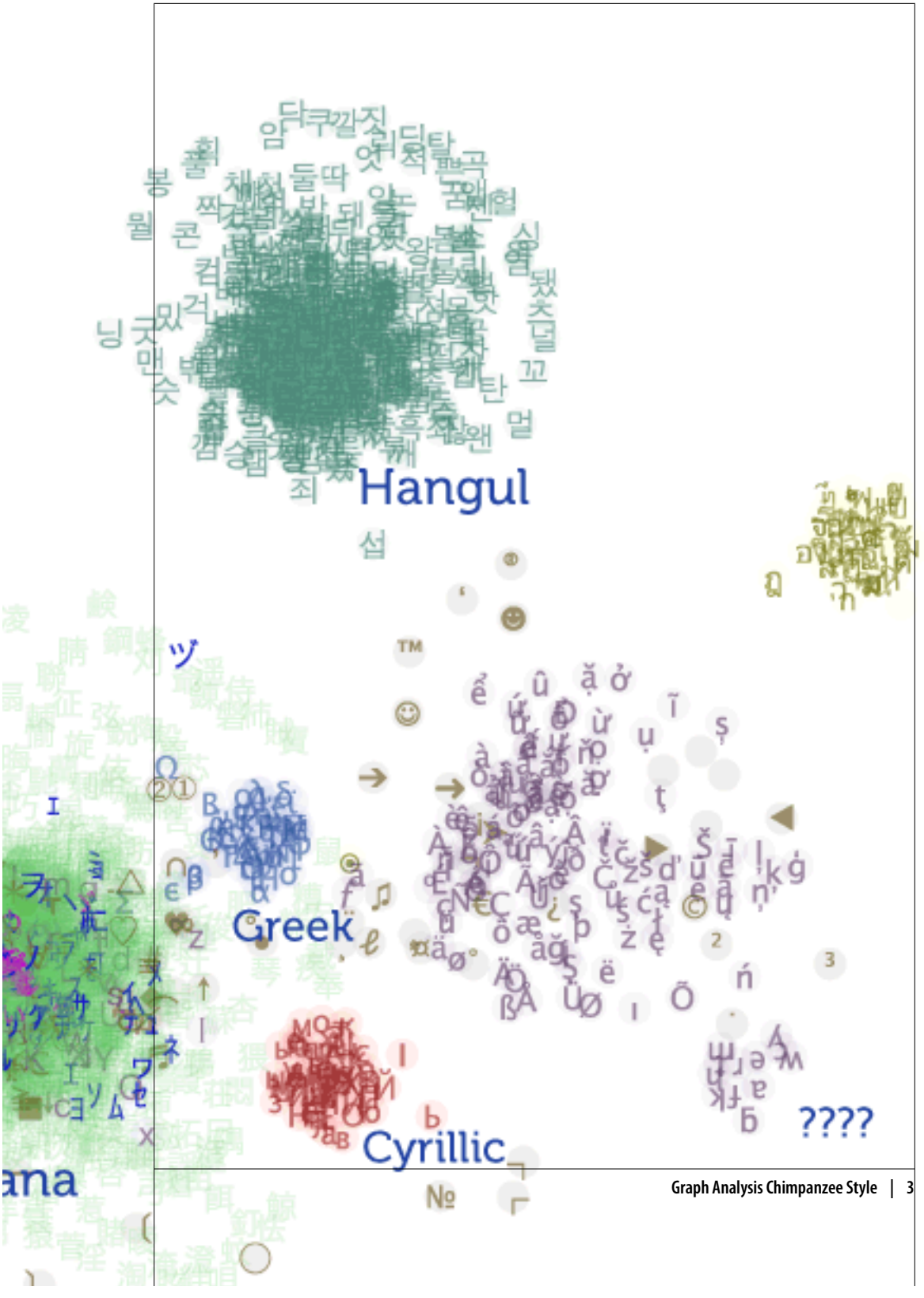
- Who are the people they converse with (the @reply graph)?
- Do the people they engage with reciprocate that attention (symmetric links)?

‡ *http://www.wormatlas.org/neuronalwiring.html*

§ *http://www.openstreetmap.org/*

‖ All are steady-state network flow problems. A flowing crowd of websurfers wandering the linked-document collection will visit the most interesting pages the most often. The transfer of social capital implied by social network interactions highlights the most central actors within each community. The year-to-year progress of students to higher or lower test scores implies what each school's effect on a generic class would be.

#Chosen without apology, in keeping with the ego-centered ethos of social networks.


Hangul
Thai
Greek
Cyrillic
????
ana

Among the user's community, how many engage with each other (clustering coefficient)?

Everybody's Talkin' At Me: The Twitter Reply Graph

Twitter lets you reply to another user's message and thus engage in conversation. Since it's an expressly public activity, a reply is a strong *social token*: it shows interest in what the other is saying, and demonstrates that interest is worth re-broadcasting.

The first step in our processing is done in Wukong, a Ruby language library for Hadoop. It lets us write small, agile programs capable of handling multi-terabyte data streams. Here is a snippet from the class that represents a twitter message (or *tweet*):*

```
class Tweet < Struct.new(:tweet_id, :screen_name, :created_at,
                         :reply_tweet_id, :reply_screen_name, :text)
  def initialize(raw_tweet)
    # ... gory details of parsing raw tweet omitted
  end

  # Tweet is a reply if there's something in the reply_tweet_id slot
  def is_reply?
    not reply_tweet_id.blank?
  true
end
```

Twitter's Stream API lets anyone easily pull gigabytes of messages†. They arrive in a raw JSON format:

```
{"text":"Just finished the final draft for Hadoop: the Definitive Guide!",
 "screen_name":"tom_e_white","reply_screen_name":null,"id":3239897342,"reply_tweet_id":null,...}
{"text":"@tom_e_white Can't wait to get a copy!",
 "screen_name":"mrflip","reply_screen_name":"tom_e_white","id":3239873453,"reply_tweet_id":3239897342,...}
{"text":"@josephkelly great job on the #InfoChimps API.
  Remind me to tell you about the time a baboon broke into our house.",
 "screen_name":"wattsteve","reply_screen_name":"josephkelly","id":16434069252,...}
{"text":"@mza Re: http://j.mp/atbroxmr Check out @James_Rubino's
  http://bit.ly/clusterfork ? Lots of good hadoop refs there too",
 "screen_name":"mrflip","reply_screen_name":"@mza","id":7809927173,...}
{"text":"@tlipcon divide lots of data into little parts. Magic software gnomes
  fix up the parts, elves then assemble those into whole things #hadoop",
 "screen_name":"nealrichter","reply_screen_name":"tlipcon","id":4491069515,...}
```

The `reply_screen_name` and `reply_tweet_id` let you follow the conversation (as you can see, they're otherwise `null`). Let's find each reply and emit the respective user IDs as an edge. ‡

```
class ReplyGraphMapper < LineStreamer
  def process(raw_tweet)
```

* You can find full working source code on the book's website.

† Refer to the Twitter developer site (*http://dev.twitter.com*) or use a tool like Hayes Davis' Flamingo (*http://github.com/hayesdavis/flamingo*).

```
    tweet = Tweet.new(raw_tweet)
    if tweet.is_reply?
      emit [tweet.screen_name, tweet.reply_screen_name]
    end
  end
end
```

The mapper derives from `LineStreamer`, a class that feeds each line as a single record to its `process` method. We only have to define that `process` method; Wukong and Hadoop take care of the rest. In this case, we use the raw JSON record to create a tweet object. Where user A replies to user B, emit the edge as A and B separated by a tab. The raw output will look like this:

```
% reply_graph_mapper --run raw_tweets.json a_replies_b.tsv
mrflip          tom_e_white
wattsteve       josephkelly
mrflip          mza
nealrichter     tlipcon
```

You should read this as "a replies b", and interpret it as a directed "out" edge: `@wattsteve` conveys social capital to `@josephkelly`.

Edge pairs vs. Adjacency List

That is the *edge pairs* representation of a network. It's simple, and it gives an equal jumping-off point for in- or out- edges, but there's some duplication of data. You can tell the same story from the node's point of view (and save some disk space) by rolling up on the source node. We call this the *adjacency list*, and it can be generated in Pig by a simple GROUP BY. Load the file:

```
a_replies_b = LOAD 'a_replies_b.tsv' AS (src:chararray, dest:chararray);
```

Then find all edges out from each node by grouping on source:

```
replies_out = GROUP a_replies_b BY src;
DUMP replies_out
```

```
(cutting,{(tom_e_white)})
(josephkelly,{(wattsteve)})
(mikeolson,{(LusciousPear),(kevinweil),(LusciousPear),(tlipcon)})
(mndoci,{(mrflip),(peteskomoroch),(LusciousPear),(mrflip)})
(mrflip,{(LusciousPear),(mndoci),(mndoci),(esammer),(ogrisel),(esammer),(wattsteve)})
(peteskomoroch,{(CMastication),(esammer),(DataJunkie),(mndoci),(nealrichter),...
(tlipcon,{(LusciousPear),(LusciousPear),(nealrichter),(mrflip),(kevinweil)})
(tom_e_white,{(mrflip),(lenbust)})
```

‡ In practice, we of course use numeric IDs and not screen names, but it's easier to follow along with screen names. In order to keep the graph-theory discussion general, I'm going to play loose with some details and leave out various janitorial details of loading and running.

Degree

A simple, useful measure of influence is the number of replies a user receives. In graph terms this is the *degree* (specifically the *in-degree*, since this is a directed graph).

Pig's nested FOREACH syntax lets us count the distinct incoming repliers (neighbor nodes) and the total incoming replies in one pass:§

```
a_replies_b = LOAD 'a_replies_b.tsv' AS (src:chararray, dest:chararray);
replies_in  = GROUP a_replies_b BY dest; -- group on dest to get in-links
replies_in_degree = FOREACH replies_in {
  nbrs = DISTINCT a_replies_b.src;
  GENERATE group, COUNT(nbrs), COUNT(a_replies_b);
};
DUMP replies_in_degree
```

```
(cutting,1L,1L)
(josephkelly,1L,1L)
(mikeolson,3L,4L)
(mndoci,3L,4L)
(mrflip,5L,9L)
(peteskomoroch,9L,18L)
(tlipcon,4L,8L)
(tom_e_white,2L,2L)
```

In this sample `@peteskomoroch` has nine neighbors and 18 incoming replies, far more than most. This large variation in degree is typical for social networks. Most users see a small number of replies, but a few celebrities—such as `@THE_REAL_SHAQ` (basketball star Shaquille O'Neill) or `@sockington` (a fictional cat)—receive millions. By contrast, almost every intersection on a road map is 4-way.‖ The skewed dataflow produced by this wild variation in degree has important ramifications for how you process such graphs—more later.

Symmetric Links

While millions of people have given `@THE_REAL_SHAQ` a shout-out on twitter, he has understandably not reciprocated with millions of replies. As the graph shows, I frequently converse with `@mndoci`#, making ours a *symmetric link*. This accurately reflects the fact that I have more in common with `@mndoci` than with `@THE_REAL_SHAQ`.

One way to find symmetric links is to take the edges in `A Replied To B` that are also in `A Replied By B`. We can do that set intersection with an inner self-join*:

§ Due to the small size of the edge pair records and a pesky Hadoop implementation detail, the mapper may spill data to disk early. If the jobtracker dashboard shows "spilled records" greatly exceeding "map output records," try bumping up the `io.sort.record.percent`:

```
PIG_OPTS="-Dio.sort.record.percent=0.25 -Dio.sort.mb=350" pig my_file.pig
```

‖ The largest outlier that comes to mind is the famous "Magic Roundabout" in Swindon, England, with degree 10, *http://en.wikipedia.org/wiki/Magic_Roundabout_%28Swindon%29*.

Deepak Singh, open data advocate and bizdev manager of the Amazon AWS cloud.

```
a_repl_to_b = LOAD 'a_replies_b.tsv' AS (user_a:chararray, user_b:chararray);
a_repl_by_b = LOAD 'a_replies_b.tsv' AS (user_b:chararray, user_a:chararray);
-- symmetric edges appear in both sets
a_symm_b_j  = JOIN a_repl_to_b BY (user_a, user_b),
                   a_repl_by_b BY (user_a, user_b);
...
```

However, this sends two full copies of the edge-pairs list to the reduce phase, doubling the memory required. We can do better by noticing that from a node's point of view, a symmetric link is equivalent to a paired edge: one out and one in. Make the graph undirected by putting the node with lowest sort order in the first slot—but preserve the direction as a piece of edge metadata:

```
a_replies_b = LOAD 'a_replies_b.tsv' AS (src:chararray, dest:chararray);
a_b_rels = FOREACH a_replies_b GENERATE
  ((src <= dest) ? src  : dest) AS user_a,
  ((src <= dest) ? dest : src)  AS user_b,
  ((src <= dest) ? 1 : 0)       AS a_re_b:int,
  ((src <= dest) ? 0 : 1)       AS b_re_a:int;
DUMP a_b_rels
```

```
(mrflip,tom_e_white,1,0)
(josephkelly,wattsteve,0,1)
(mrflip,mza,1,0)
(nealrichter,tlipcon,0,1)
```

Now gather all edges for each node pair. A symmetric edge has at least one reply in each direction:

```
a_b_rels_g   = GROUP a_b_rels BY (user_a, user_b);
a_symm_b_all = FOREACH a_b_rels_g GENERATE
  group.user_a AS user_a,
  group.user_b AS user_b,
  (( (SUM(a_b_rels.a_re_b) > 0) AND
     (SUM(a_b_rels.b_re_a) > 0)     ) ? 1 : 0) AS is_symmetric:int;
DUMP a_symm_b_all
```

```
(mrflip,tom_e_white,1)
(mrflip,mza,0)
(josephkelly,wattsteve,0)
(nealrichter,tlipcon,1)
...
```

```
a_symm_b = FILTER a_symm_b_all BY (is_symmetric == 1);
STORE a_symm_b INTO 'a_symm_b.tsv';
```

Here's a portion of the output, showing that `@mrflip` and `@tom_e_white` have a symmetric link:

```
(mrflip,tom_e_white,1)
(nealrichter,tlipcon,1)
...
```

* Current versions of Pig get confused on self-joins, so just load the table with differently-named relations as shown here.

Community Extraction

So far we've generated a node measure (in-degree) and an edge measure (symmetric link identification). Let's move out one step and look at a neighborhood measure: how many of a given person's friends are friends with each other? Along the way, we'll produce the edge set for a visualization like the one above.

Get neighbors

Choose a seed node (here, `@tom_e_white`). First round up the seed's neighbors:

```
a_replies_b = LOAD 'a_replies_b.tsv' AS (src:chararray, dest:chararray);
-- Extract edges that originate or terminate on the seed
n0_edges   = FILTER a_replies_b BY (src == 'hadoop') OR (dest == 'hadoop');
-- Choose the node in each pair that *isn't* our seed:
n1_nodes_all = FOREACH n0_edges GENERATE
  ((src == 'hadoop') ? dest : src) AS screen_name;
n1_nodes    = DISTINCT n1_nodes_all;
DUMP n1_nodes
```

Now intersect the set of neighbors with the set of starting nodes to find all edges originating in `n1_nodes`:

```
n1_edges_out_j = JOIN a_replies_b BY src,
                      n1_nodes    BY screen_name USING 'replicated';
n1_edges_out   = FOREACH n1_edges_out_j GENERATE src, dest;
```

Our copy of the graph (with more than 1 billion edges) is far too large to fit in memory. On the other hand, the neighbor count for a single user rarely exceeds a couple million, which fits easily in memory. Including `USING 'replicated'` in the JOIN command instructs Pig to do a map-side join (also called a *fragment replicate join*). Pig holds the `n1_nodes` relation in memory as a lookup table, and streams the full edge list past. Whenever the join condition is met—`src` is in the `n1_nodes` lookup table—it produces output. No reduce step means an enormous speedup!

To leave only edges where both source and destination are neighbors of the seed node, repeat the join:

```
n1_edges_j = JOIN n1_edges_out BY dest,
                  n1_nodes     BY screen_name USING 'replicated';
n1_edges   = FOREACH n1_edges_j GENERATE src, dest;
DUMP n1_edges
```

```
(mrflip,tom_e_white)
(mrflip,mza)
(wattsteve,josephkelly)
(nealrichter,tlipcon)
(bradfordcross,lusciouspear)
(mrflip,jeromatron)
(mndoci,mrflip)
(nealrichter,datajunkie)
```

Community metrics and the 1 million x 1 million problem

With `@hadoop`, `@cloudera` and `@infochimps` as seeds, I applied similar scripts to 2 billion messages to create Figure 17-2.

As you can see, the big data community is very interconnected. The link neighborhood of a celebrity such as `@THE_REAL_SHAQ` is far more sparse. We can characterize this using the *clustering coefficient*: the ratio of actual `n1_edges` to the maximum number of possible `n1_edges`. It ranges from zero (no neighbor links to any other neighbor) to one (every neighbor links to every other neighbor). A moderately high clustering coefficient indicates a cohesive community. A low clustering coefficient could indicate widely-dispersed interest (as it does with `@THE_REAL_SHAQ`), or it could indicate the kind of inorganic community that a spam account would engender.

Local Properties at Global Scale

We've calculated community metrics at the scale of a node, an edge, and a neighborhood. How about the whole globe? There's not enough space here to cover it, but you can simultaneously determine the clustering coefficient for every node by generating every "triangle" in the graph. For each user, comparing the number of triangles they belong to with their degree leads to the clustering coefficient.

Be careful, though! Remember the wide variation in node degree discussed above? Recklessly extending the previous method will lead to an explosion of data—pop star `@britneyspears` (5.2M followers, 420k following as of July 2010) or `@WholeFoods` (1.7M followers, 600k following) will each generate trillions of entries. What's worse, since large communities have a sparse clustering coefficient almost all of these will be thrown away! There is a very elegant way to do this on the full graph†, but always keep in mind what the real world says about the problem. If you're willing to assert that `@britney spears` isn't *really* friends with 420,000 people, you can keep only the strong links. Weight each edge (by number of replies, whether it's symmetric, and so on) and set limits on the number of links from any node. This sharply reduces the intermediate data size yet still does a reasonable job of estimating cohesiveness.

—Philip (flip) Kromer, Infochimps

† See *http://www.slideshare.net/ydn/3-xxl-graphalgohadoopsummit2010*—Sergei Vassilvitskii (`@vsergei`) and Jake Hofman (`@jakehofman`) of Yahoo! Research solve several graph problems by very intelligently throwing away most of the graph.


flowingdata
joshdilworth
damon
infochimps
novaspivack
mrflip
znmeb
stephenfry
marshallk
chrismessina
nytimes
rww
pkedrosky
timoreilly
davemcclure
monkchips
al3x
gigaom
mattcutts
newsycombinator
alisonhwerner
igrigorik
jeffbarr
kevinweil
LusciousPear
nosqlupdate
cloudera
spyced
debasishg
komoroch
juansequeda
caseorganic
Pistachio
karllong
danmartell
Jason
randybias
jamesurquhart