Connecting the Dots in Wikipedia

Tidewater Big Data Enthusiasts
Chuck Cartledge
Developer

April 23, 2020 at 12:45 Noon

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Dirk Gently's code is that everything is connected[1]. This the basis of his holistic detective agency, and is his life's guiding principle. We will take Dirk's principal idea and apply it to Wikipedia. We'll explore a portion of Wikipedia and see how many pages/links separate a specific page from another (in graph theory this is called the shortest path, Stanley Milgram made this idea popular with his "Small-World" paper, and many people have played the "Six Degrees of Kevin Bacon" game). We'll also look at the average number of links between pages.

Along the way, we'll look at words that are common between the pages, the educational level of the pages, and other things if we have time. For fun; we'll use SQL and non-SQL databases to manage the exploration process and keep track of the results.

2 Discussion

Don't do this at home. Or anywhere else.

Exploring the Wikipedia graph is a common enough activity (see Figure 1 on the following page) that Wikipedia provides a downloadable XML file that represents their graph[1] (see Figure 2 on page 3). Downloading and processing an XML file that includes the totality of Wikipedia might be a more productive effort than writing a crawler to follow page links. Wikipedia makes their data available for free, and specifically requests that they not be crawled by bots or scrapers. All that being said; how did we do, and what did we find out?

We'll start with Albert Einstein's Wikipedia page (because we have to start somewhere) (see Figure 3 on page 4). A quick estimate of the number of links from this page is about 1,000 based on the number of "pages" in the browser, and multiplying that by an average number of links per page. Looking at the HTML source for the page (see Figure 4 on page 5), copying the HTML to a editor, and replacing all "href" tags with "href" returns 2,774 changes.

Wikipedia provides additional ways to poke and prod their system, including an API[2] (see Figure 5 on page 6). In particular, we will use the page properties API[3] because it will allow us to search for a page, and then "page" through the properties on the Wikipedia page.

2.1 System Overview

While the system is fairly easy to talk about, it helps to have a diagram that shows how the various "bits and pieces" fit together (see Figure 7 on page 9). There are three major areas where things happen. In the purple area are R scripts that download data from Wikipedia,

[1] `https://en.wikipedia.org/wiki/Wikipedia:Database_download`

[2] `https://www.mediawiki.org/wiki/API:Properties`

[3] `https://www.mediawiki.org/wiki/API:Pageprops`

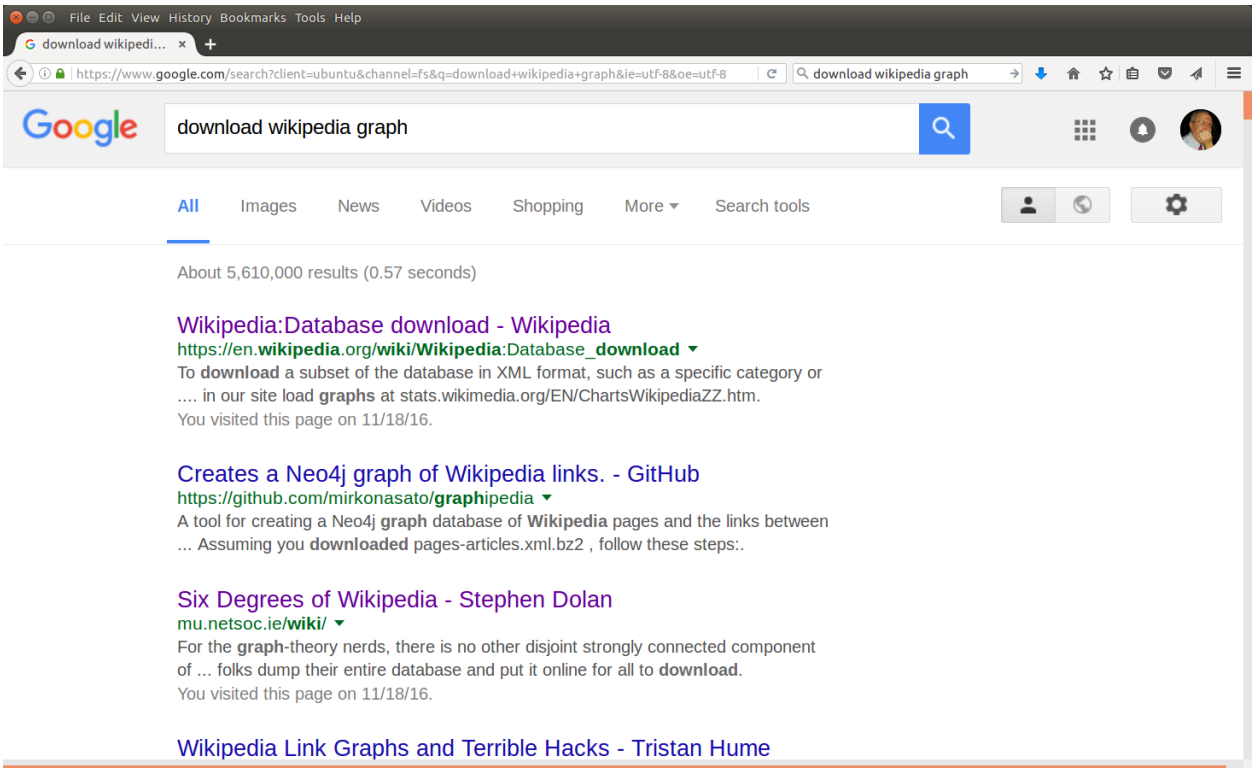

Figure 1: Searching for Wikipedia graph data. More than 5,610,000 pages were identified using the terms: download Wikipedia graph. The number of pages can be indicator of how "common" this effort/exercise is.

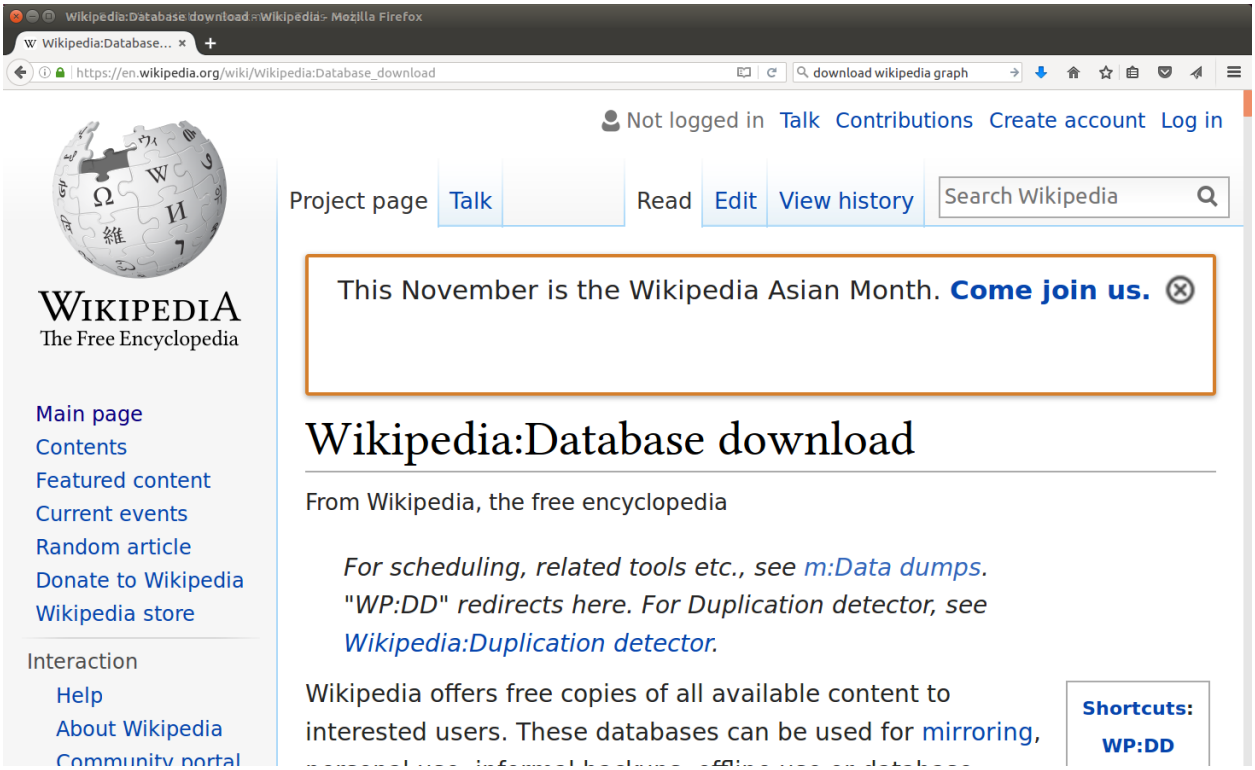

Figure 2: Wikipedia graph download page.

Figure 3: Albert Einstein's Wikipedia page. URL: `https://en.wikipedia.org/wiki/Albert_Einstein`

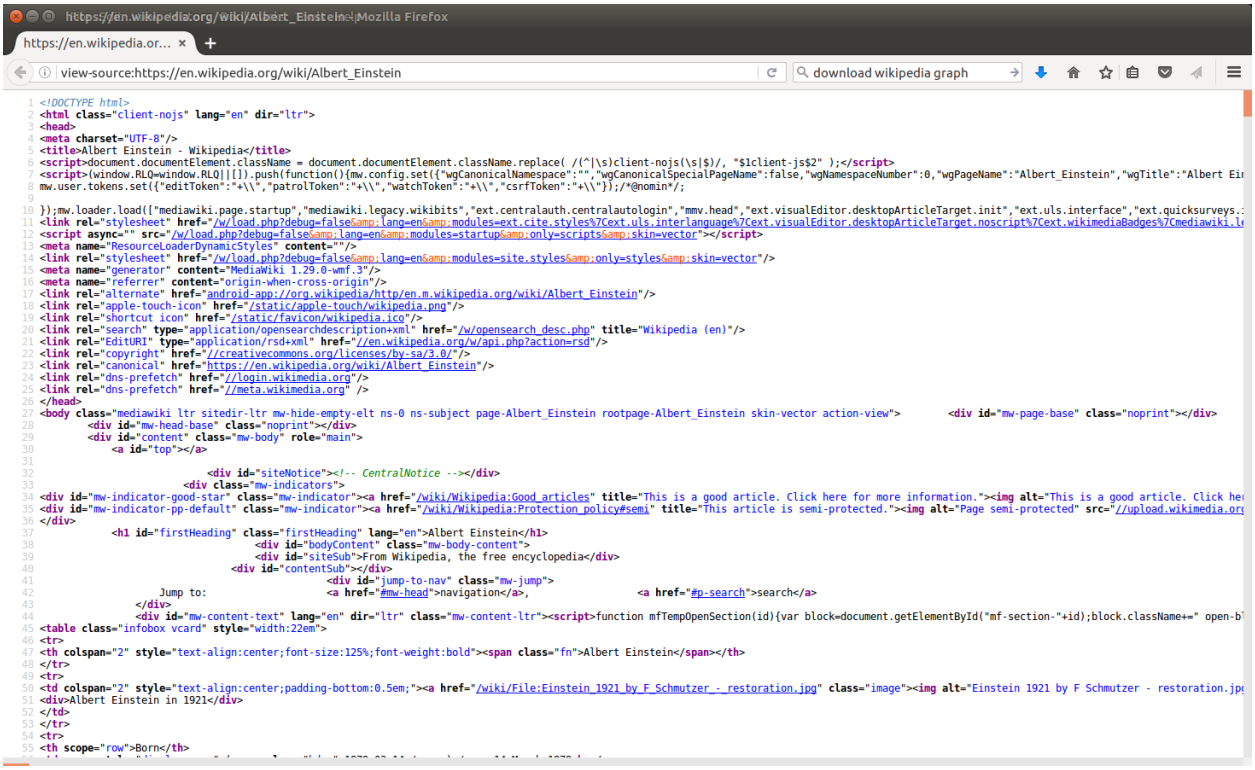

Figure 4: Source of Albert Einstein's Wikipedia page.

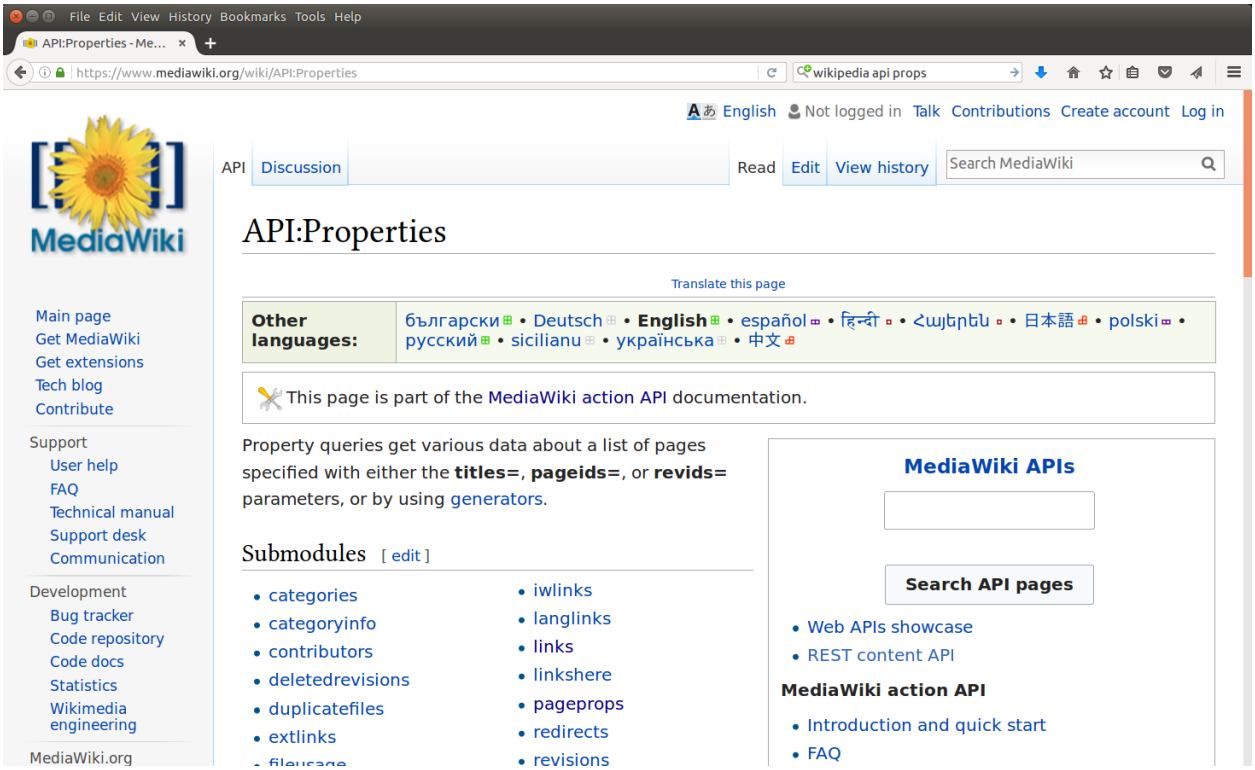

Figure 5: Wikipedia API page. URL:`https://www.mediawiki.org/wiki/API:Properties`

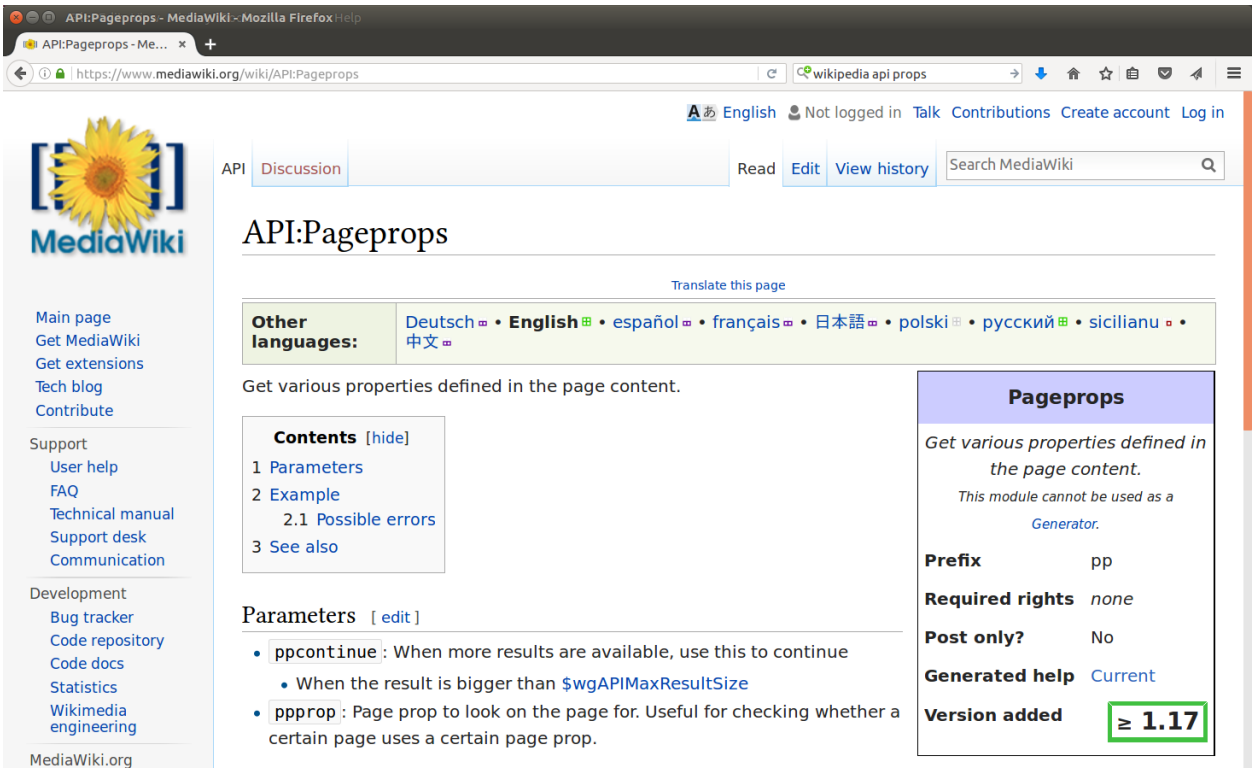

Figure 6: Wikipedia API page properties page. URL:`https://www.mediawiki.org/wiki/API:Properties`

produce the various plots in this report, and finally explore the Neo4j database. The green area indicates the PostgreSQL tables. And, the yellow are are activities carried out in a Makefile.

The wikipedia R script retrieves pages from Wikipedia and updates various PostgreSQL tables. Data from these tables is extracted, transformed, and loaded into the Neo4j database. A final R script explores the Neo4j database looking to solve problems and queries.

2.2 PostgreSQL database

A PostgreSQL database was chosen to store the Wikipedia graph as it was explored, and to coordinate activities of the different simultaneous explorations. PostgreSQL' ACID properties (Atomicity, Consistency, Isolation, and Durability) would help to orchestrate the activities of the explorers. The ToDO and Done tables (see Table 1 on page 10) record the work to be done, and what has been done. The Graph table (see Table 2 on page 11) maintains all the arcs that have been discovered so far. If the collective actions of the explorers are not managed globally, they could continue forever. The Limits table (see Table 3 on page 12) keeps the total number of pages that have been processed during any particular exploration effort. Each explorer updates this table when they have completed processing a page, and checks the table's value against the total number to be processed. If the number from the Limits table is less than the directed maximum, then more pages are explored.

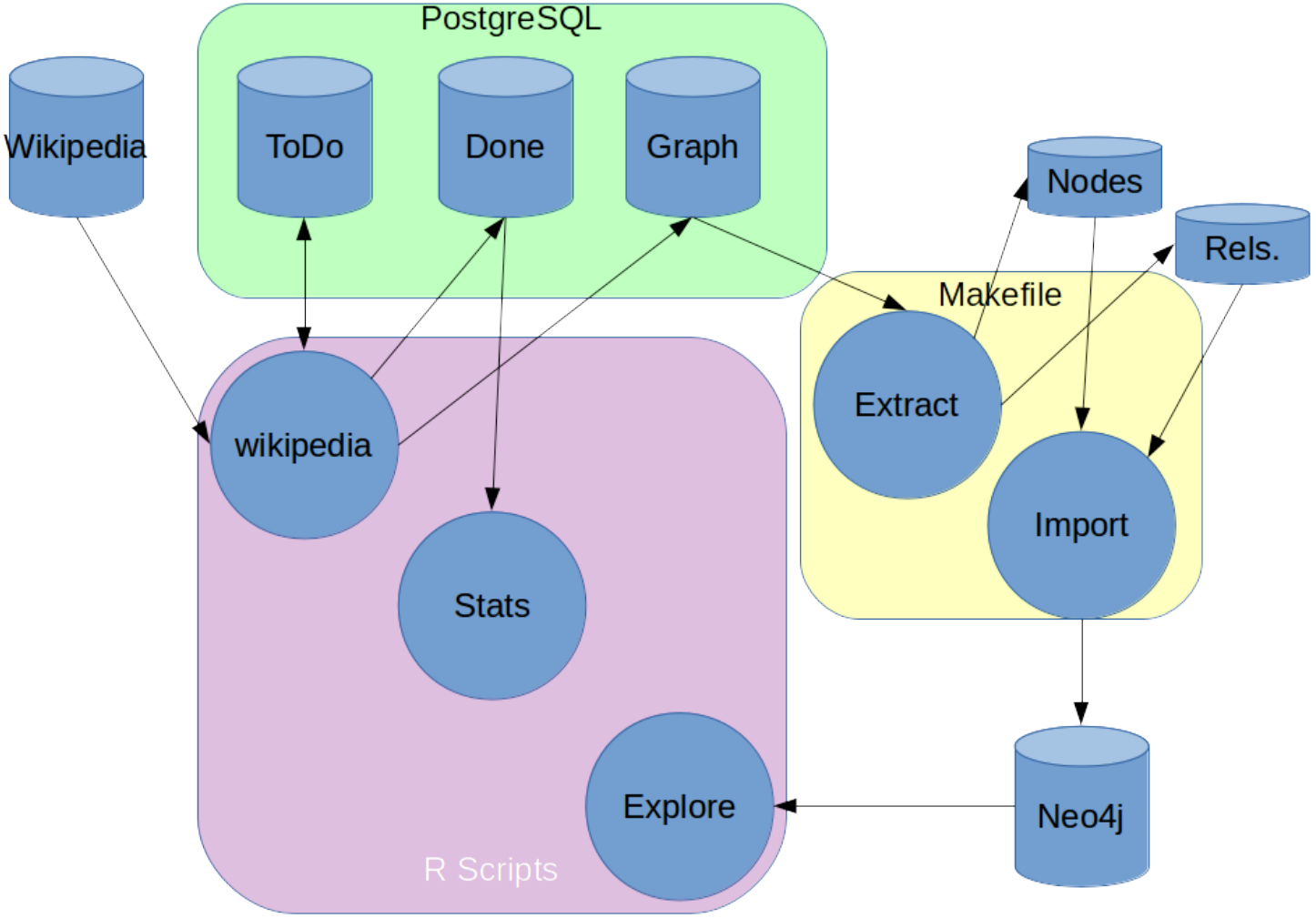

Figure 7: Overall system performance diagram.

Table 1: The PostgreSQL ToDo and Done tables.

Col. name	**Type**	**Explanation**
title	character varying(8000)	Base 64 encoded of the Wikipedia title. Primary key.
node	integer	The node (explorer) assigned to process this page.
added	integer	Unix seconds when this page was added to the work queue.
started	integer	Unix seconds when the node started to process this page.
finished	integer	Unix seconds when the node finished processing this page.
languished	integer	Unix seconds between the time when the page was added to the queue and when the node started to process the page.
processed	integer	Number of seconds to process the page.
connections	integer	Number of outgoing links in the page.

Table 2: The Graph PostgreSQL table.

Col. name	**Type**	**Explanation**
id	serial	Primary key.
sourcenode	character varying (8000)	The page with the out going connection.
destinationnode	character varying (8000)	The page that is connected to by the source.

Table 3: The Limits PostgreSQL table.

Col. name	**Type**	**Explanation**
limits	integer	The total number of pages currently explored by all explorers.

2.3 Neo4j database

After the PostgreSQL tables are populated by the explorers, the tables are processed and fed into a Neo4j database. Originally updating and creating the Neo4j was to be done while exploring Wikipedia. It became rapidly apparent that this was slowing the whole process down to the point of being unreasonable. The approach finally used was:

1. Have the explorers populate the PostgreSQL database (see wikipedia.R in Section A on page 37),
2. Stop the Neo4j database (see Makefile in Section A on page 37),
3. Extract and reformat PostgreSQL data into Neo4j import files,
4. Use the Neo4j-import program to create a Neo4j database, and
5. Start the Neo4j database.

First few lines from the `nodes.csv` file:

```
nodeID:ID,title,:LABEL
0a4=,0a4=,Page
0bI=,0bI=,Page
0bQ=,0bQ=,Page
0I4=,0I4=,Page
0I8=,0I8=,Page
0IbQu9GM0L3QuNGG0Y8=,0IbQu9GM0L3QuNGG0Y8=,Page
0Ic=,0Ic=,Page
```

First few lines from the `rels.csv` file:

```
:START_ID,:END_ID,:TYPE
QWdhc3R5YQ==,Ui4gUmFnaGF2YSBJeWVuZ2Fy,CONNECTED_TO
QWdhc3R5YQ==,UmFnaGF2YSBJeWVuZ2Fy,CONNECTED_TO
QWdhc3R5YQ==,UmFpa3Zh,CONNECTED_TO
QWdhc3R5YQ==,UmFqYXJzaGk=,CONNECTED_TO
QWdhc3R5YQ==,UmFqYXNla2hhcmE=,CONNECTED_TO
QWdhc3R5YQ==,UmFtYQ==,CONNECTED_TO
QWdhc3R5YQ==,UmFtYXlhbmE=,CONNECTED_TO
```

The `nodeID:ID` values **must** be unique in the graph, because they are the IDs used by Neo4j for searching, matching, and other operations. Duplicate IDs will cause the import process to fail. `nodes.psql` contains the relatively simple (but long running) select statement to get unique values.[4] Processing of the relationship file (`rels.csv`) is more relaxed. If

[4]It might be just as fast to do a simple psql select on the Graph table, then used external programs to split the output into two columns that are appended together, sorted, and then unique values used to populate the `nodes.csv` file.

either the `:START_ID` or `:END_ID` is not found, then the line is skipped. Missing `:START_ID` or `:END_ID` does not cause an error.

2.4 Algorithms

The 50,000 view of an exploring process is very simple (see Algorithm **??** on page ??). Algorithmicly, the system can support an unlimited number of explorers. From a practical sense, each explorer consumes about 70K of RAM when running so the amount of available RAM will limit the number of explorers that should be run simultaneously. Each explorer has the same set of command line arguments (see Table 4 on the following page).

As with any parallel operating system, there needs to be a controlling process. With this system, that responsibility falls on node 0 (see Algorithm 1 on page 16). After node 0 completes it's special processing, it continues operating like all other nodes.

Management of the ToDo and Done tables

The management of the ToDo, and Done tables for process management is not optimal. Each page is assigned to a node based on the individual node assignment value maintained by the individual explorer nodes. It is possible that the same page could be assigned to multiple nodes. If the page has been completed, as recorded in the Done table, then the page is not processed regardless of which node completed it. An error condition can occur when the same page is entered into the ToDo table when the page already exists in the table. A simular error condition can occur if the page is entered into the Done table more than once. This type of condition is managed by the R script by using the `tryCatch` structure. While these types of occurrences is real, hopefully they won't happen too often, and this is a non-mission critical program. There may be more elegant ways to handle the error condition, but it isn't worth the time and effort to implement them.

Table 4: Command line arguments. The order of the command line options is not important. If the same option appears more than once, only the last one will be used. Be aware that the **r** option does not take a value. If it is present, then it's value will be TRUE.

Letter	**Default**	**Meaning or use.**
n	0	This explorer instance number.
N	9	Number of explorer nodes.
p	1000	Maximum number of pages the system should process.
r	False	Reset the databases to empty.
s	"Albert Einstein"	Default seed page.

Algorithm 1: Special processing for node 0.

```
Data: All command line arguments
Result: Depends on command line arguments, possibly reset databases, explorer
        nodes started in the background
nodeAssignment = 0;
if my node number = 0 then
    if reset database flag is TRUE then
        reset PostgreSQL database and create tables ;
        add seed page to ToDo table;
        reset Neo4j database;
    reset the of completed pages to 0;
    for i in (number of desired nodes - 1) do
        start background nodes with n, p, and N command line arguments;
```

3 Results

3.1 Graph results

The system was directed to reset the database, start exploring Wikipedia starting at Albert Einstein's page, and process 10,000 pages (see Table 5 on the following page). Data on system performance was captured in the Done table, and was analyzed using the `wikipediaStats.R` script. Unexpectedly, the order that data was selected from the table affected the results (see Figures 8 on page 19 through 11 on page 22). While the plots seem really, really different, examination of the data (see Table 5 on the next page) shows that aside from the intercept values, the incremental change (the slope value) is very nearly identical across all plots. Histograms of time to process a page (see Figure 12 on page 23), number of out going connections (see Figure 13 on page 24), and how long pages languished before be processed (see Figure 14 on page 25) were created.

Using the idea that Wikipedia has approximately 5,000,000 pages, and that our 10,000 took approximately 11,460 seconds, then all of Wikipedia would take 5,730,000 seconds (or 66 days) to crawl using 9 crawlers.

Table 5: System behavior for 10,027 pages. The system was directed to process 10,000. Due to the way the Limits table was checked, a few more pages were processed.

Seconds	**Activity**
11,460	Explore Wikipedia.
90	Extract data from the PostgreSQL database and create Neo4j import files.
52	Stop the Neo4j database, import the new data, and start the Neo4j database.

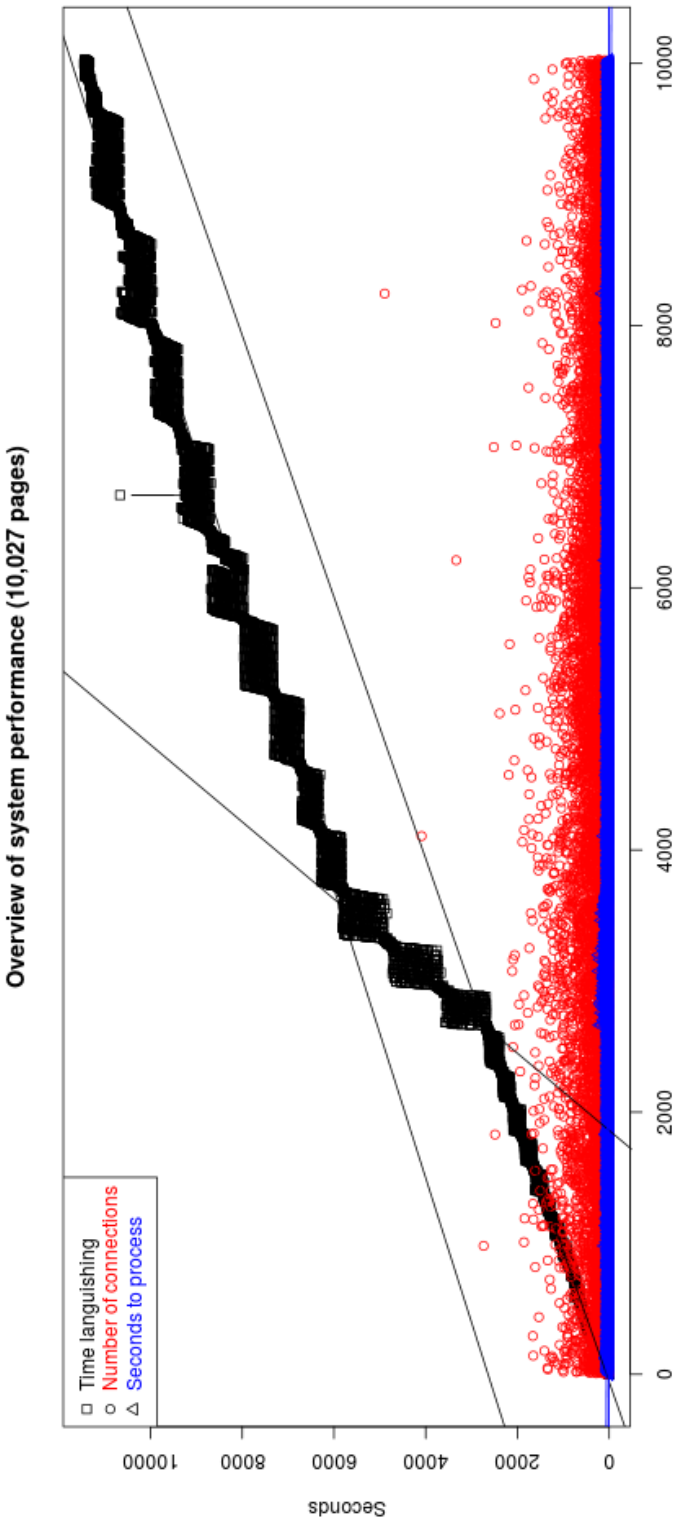

Figure 8: Overall system performance default select statement.

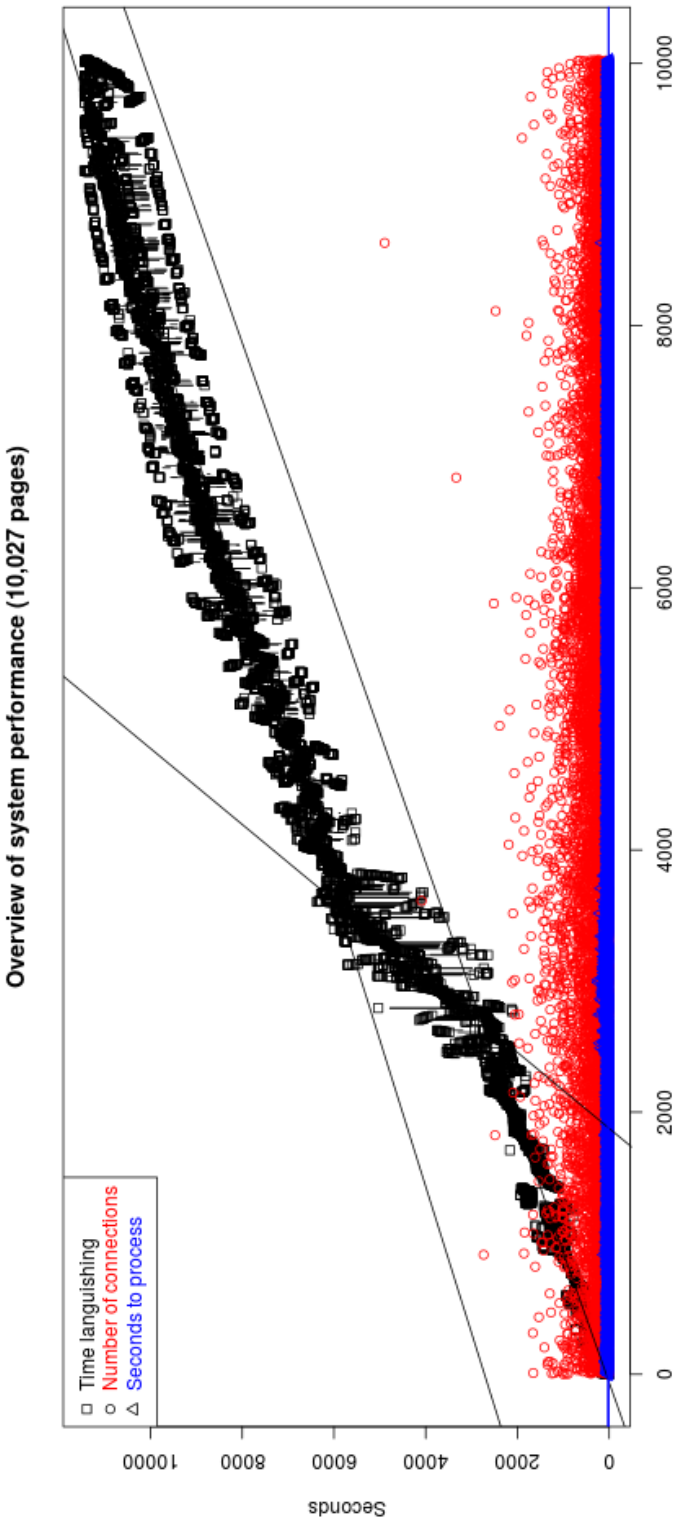

Figure 9: Overall system performance data ordered by "added".

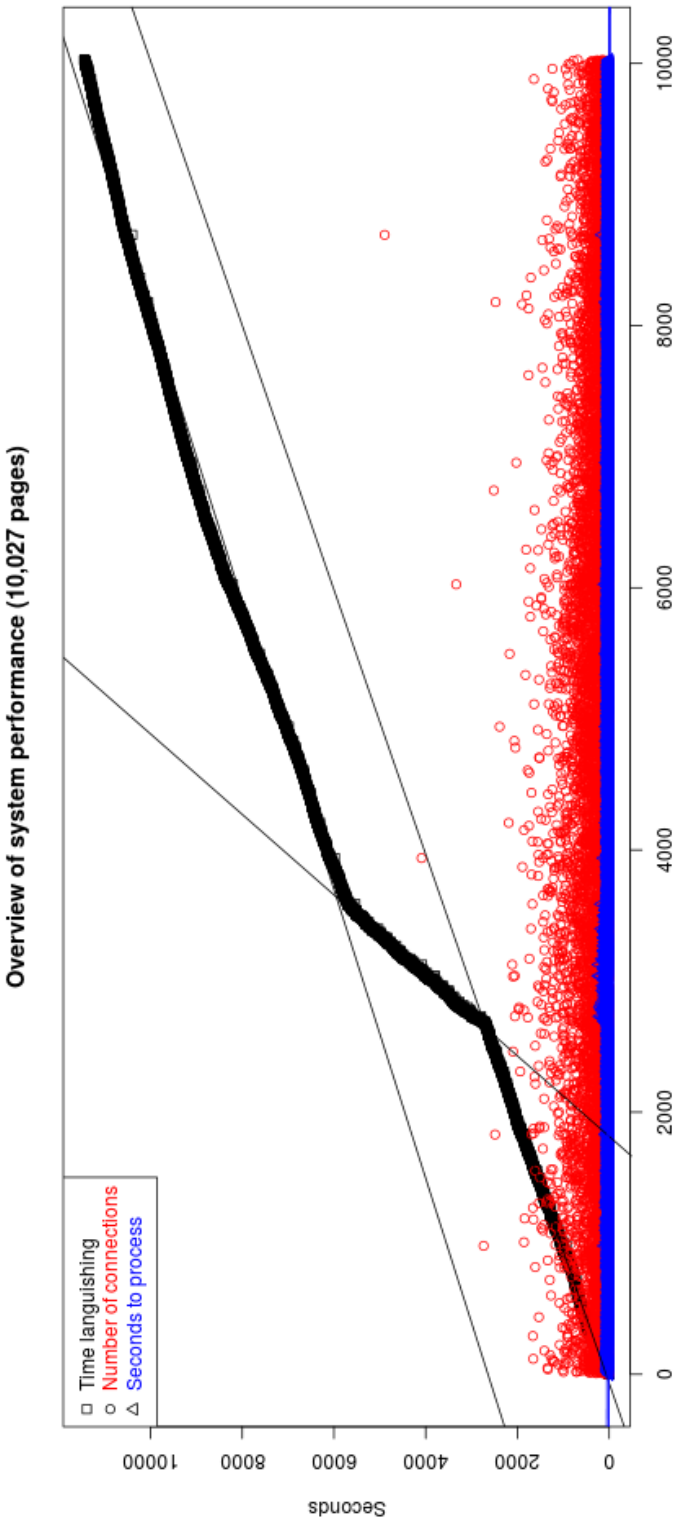

Figure 10: Overall system performance data ordered by "finished".

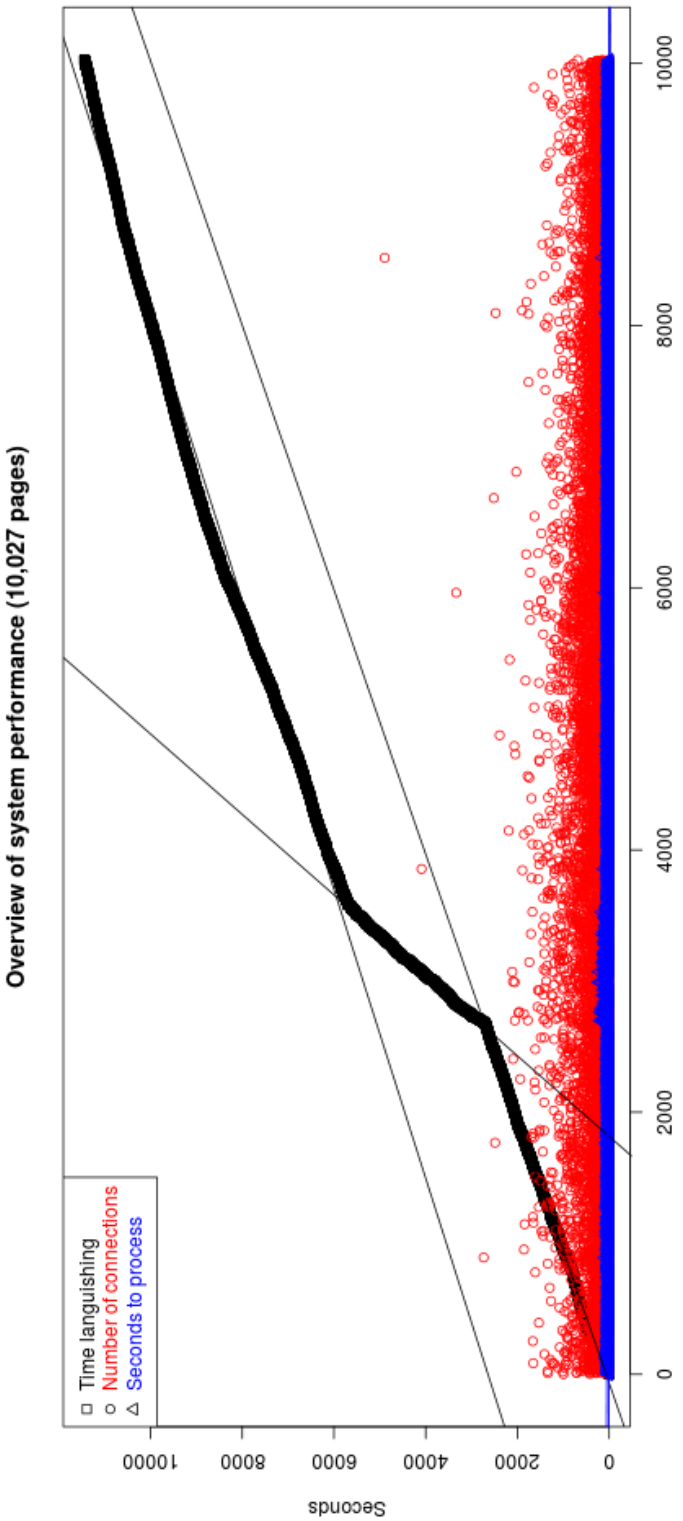

Figure 11: Overall system performance data ordered by "started".

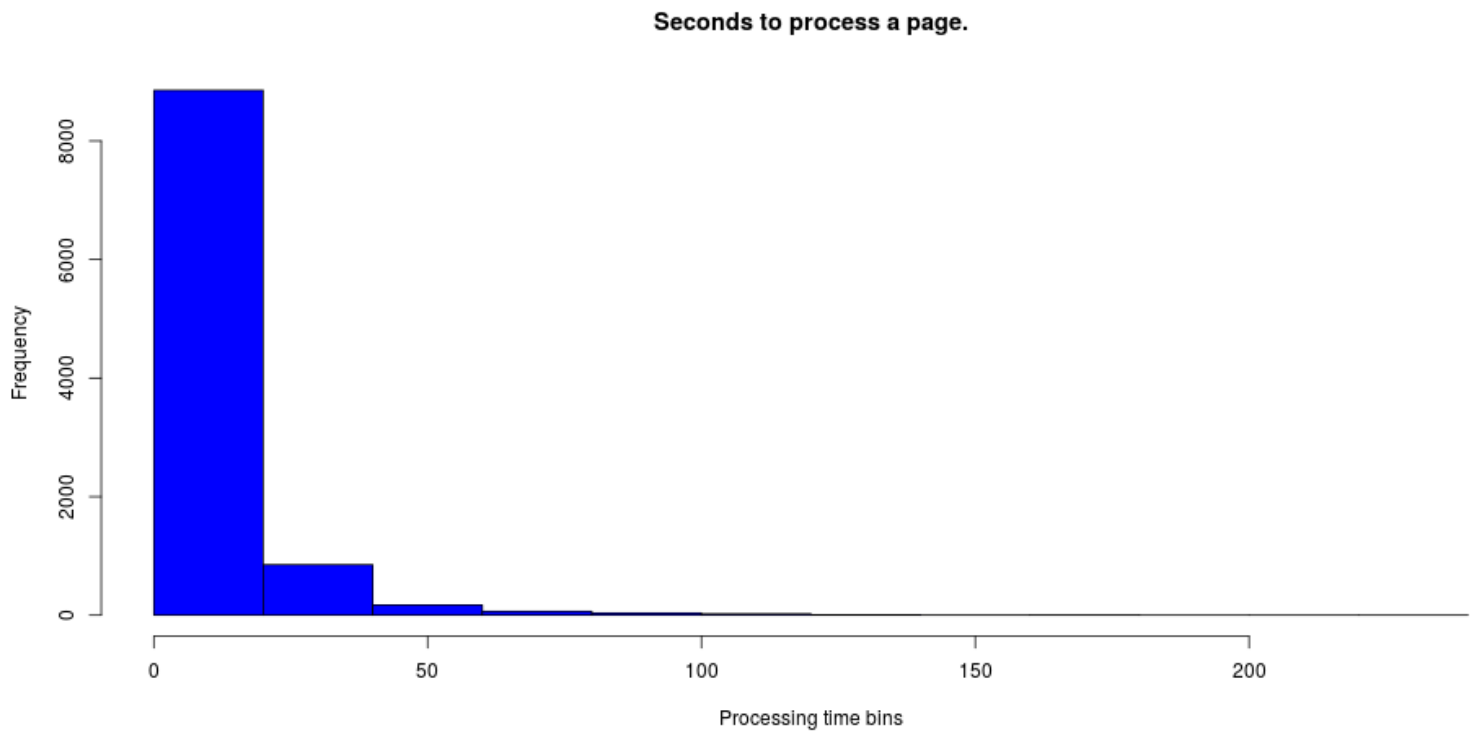

Figure 12: Histogram of time to process a page based on system performance data.

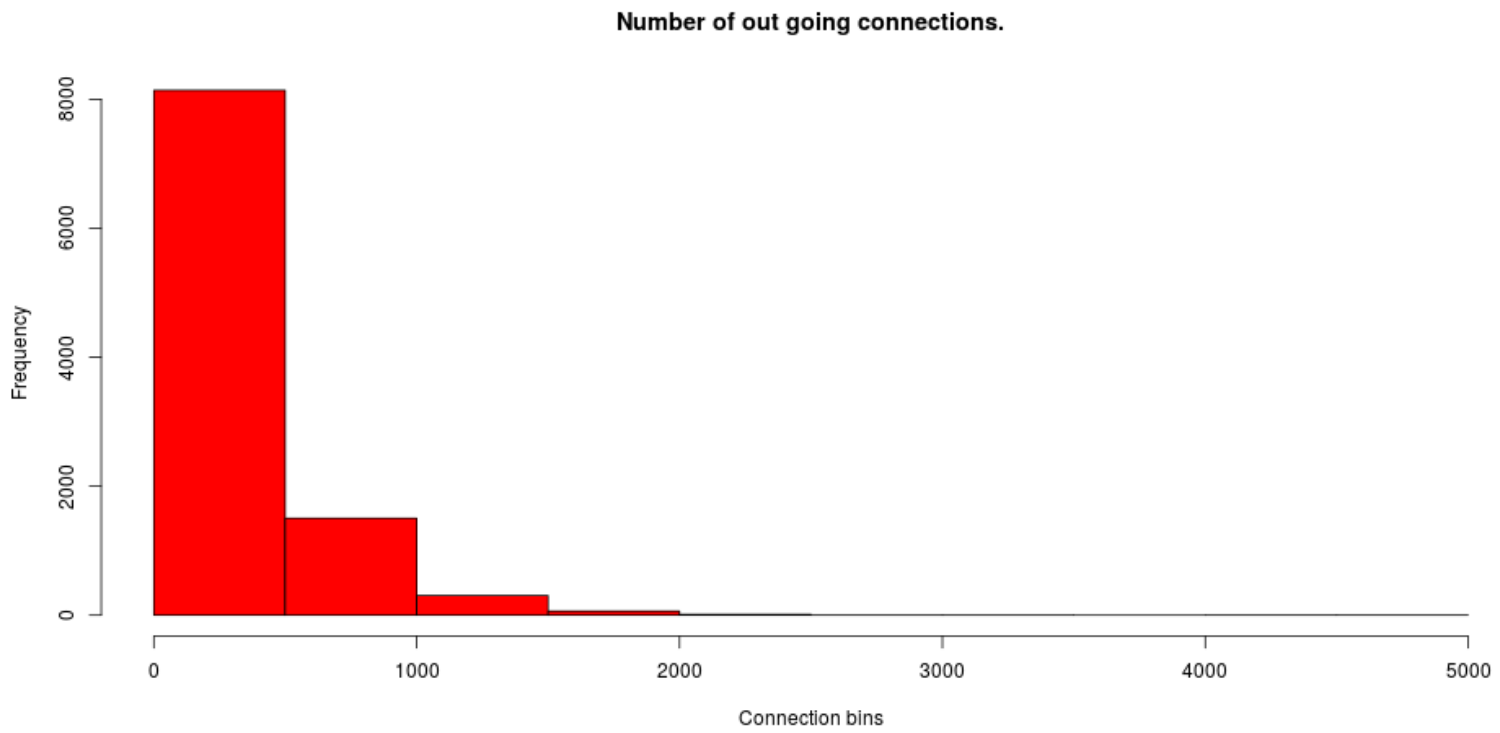

Figure 13: Histogram of out going connections based on system performance data.

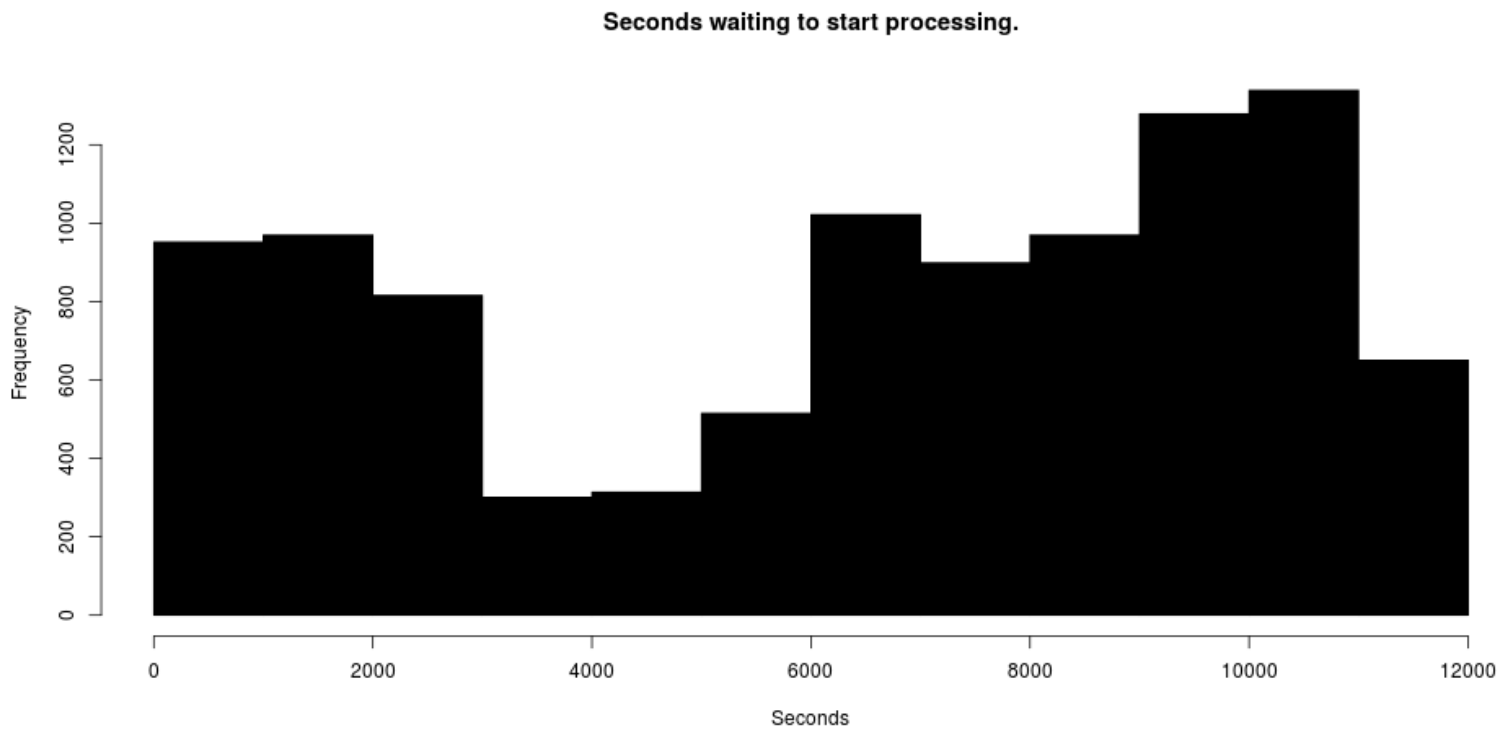

Figure 14: Histogram of languish time based on system performance data.

Table 6: Linear modeling of system performance. Examination of the system performance plots revealed three distinct areas (segments 1, 2, and 3). Linear equation coefficients for each segment for languish and processing times were computed.

		Languish		**Processing**	
Seg.		Intercept	Slope	Intercept	Slope
1	Raw	59.490919	1.001918	9.243165	0.000066
	Started	68.276183	0.991335	9.202748	0.000190
	Added	64.618835	1.008543	8.349092	0.001286
	Finished	68.306086	0.991338	9.312100	-0.000006
2	Raw	-6323.104973	3.399055	60.916314	-0.010478
	Started	-5910.560290	3.257141	55.416966	-0.008326
	Added	-6472.055024	3.450379	32.679855	-0.002093
	Finished	-5909.112869	3.256709	52.957257	-0.007515
3	Raw	2642.458391	0.907123	14.625972	-0.001034
	Started	2643.748884	0.907760	13.590724	-0.000906
	Added	2719.591729	0.894666	14.611140	-0.001037
	Finished	2643.622983	0.907775	13.724723	-0.000920

3.2 Using the graph

All of the previous results were statistics about the Wikipedia graph that was returned by the 9 explorers. Now we'll look at using the graph. The original intent was to see if was possible to connect two arbitrary Wikipedia pages together by following the links between pages. We were able to down 0.2% of the graph, so our local explorations are severely restricted. Our first attempt using the Neo4j browser on out local machine (see Figure 20 on page 35) caused the browser to hang. So `wikipediaExplore02.R`, a limited capability Neo4j explorer was created.

`wikipediaExplore02.R` uses the title for source and destination Wikipedia pages to try and find the shortest path between the two pages. Because of the small graph size, it is possible that a path does not exist the pages. The script has source and destination pages written into the code, and they can be changed as needed or necessary. Currently, the source page is "Aortoiliac occlusive disease" and the destination is "Adolf Hitler". Both the source and destination pages are checked against the current Neo4j database to see if they are there. If neither page is in the database, then no path is possible. If both pages are in the database, then a path from the source to the destination is requested that is less than 100 arcs in length. Afterwards, a reverse path from the destination to the source is requested. It is absolutely possible that there may be a path in one direction, but not the other. We've included the results of the default testing (see Table 7 on the following page).

Table 7: Connecting the dots between two pages. The between the original source and destination is different than the path from the destination to the source.

Output	**Explanation**
Path length = 3	A path was found and it has 3 edges.
Path starts (page titles).	A header.
————	A header.
Aortoiliac occlusive disease	The starting page. "...disorders of the two major blood vessels that feed the lower half of the body"
Atherosclerosis	"...a disease of the arteries characterized by the deposition of plaques of fatty material on their inner walls"
Korean War	"The Korean War began when North Korea invaded South Korea."
Adolf Hitler	"...leader of the Nazi Party and became Chancellor of Germany in 1933."
————	A trailer.
Path ends.	
	Blank line.
Source and destination nodes are swapped.	Informative statement.
	Blank line.
Path length = 3	A path was found and it has 3 edges.
Path starts (page titles).	A header.
————	A header.
Adolf Hitler	The starting page.
Tinnitus	"...is a physical condition, experienced as noises or ringing in the ears or head when no such external physical noise is present."
Atherosclerosis	
Aortoiliac occlusive disease	
————	A trailer.
Path ends.	A trailer.
The program has ended.	Informative statement.

A simple textual analysis of the text of each page along the path from the source to destination page was conducted. Stop words were removed, everything was lower cased, punctuation and numbers were removed, and all non-ASCII characters were removed. The remaining words were compared to a grade appropriate vocabulary[2]. The number of words in each grade level was plotted (see Figures **??** on page ?? through **??** on page ??). This approach is very dependent on the selection of vocabulary for each grade level, and begins to fail when excessively technical words are used. Based on the sample pages processed, the median grade level is about 5.

Oh By The Way

When constructing the `wikipediaExplore02.R` script, I had to choose source and destination pages that were in the Neo4j database. After a path was found, I went to the Wikipedia pages in the path to verify the solution. The text shown in the output from the `wikipediaExplore02.R` script are the title of Wikipedia pages, not text from the body of the page inside the HTML anchor ($< a$) tags. You have to look at the source of the page to find the page titles as returned by the script.

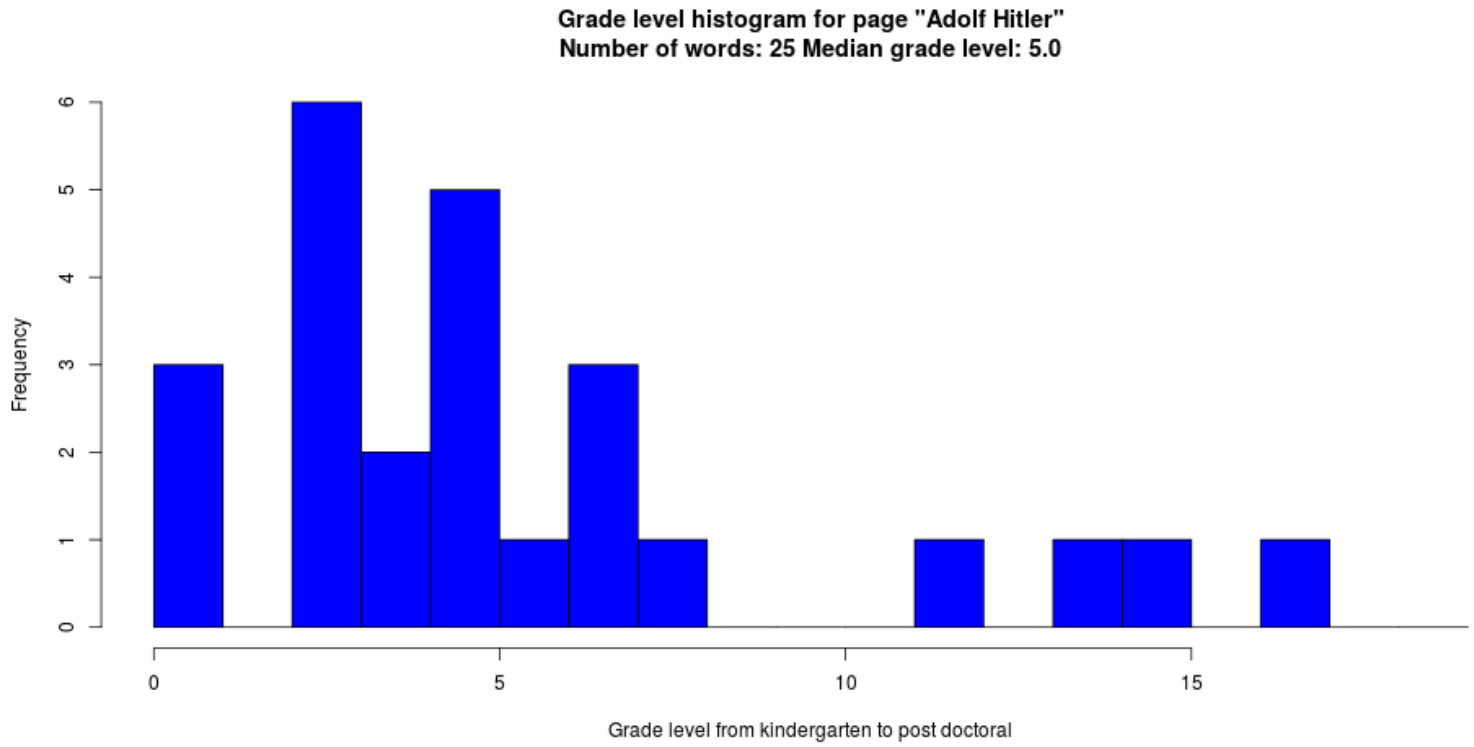

Figure 15: Page Adolf Hitler histogram of grade level vocabulary.

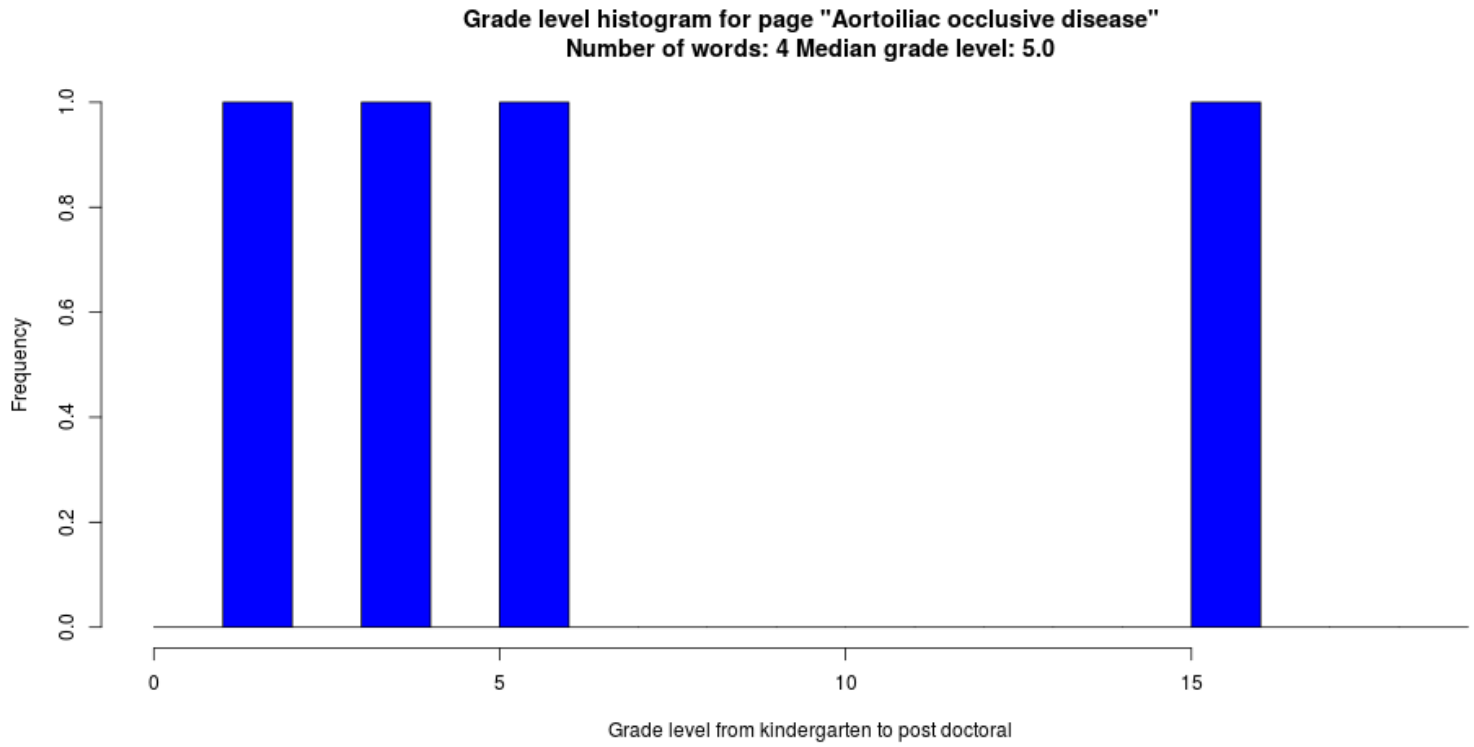

Figure 16: Page Aortoiliac occlusive disease histogram of grade level vocabulary.

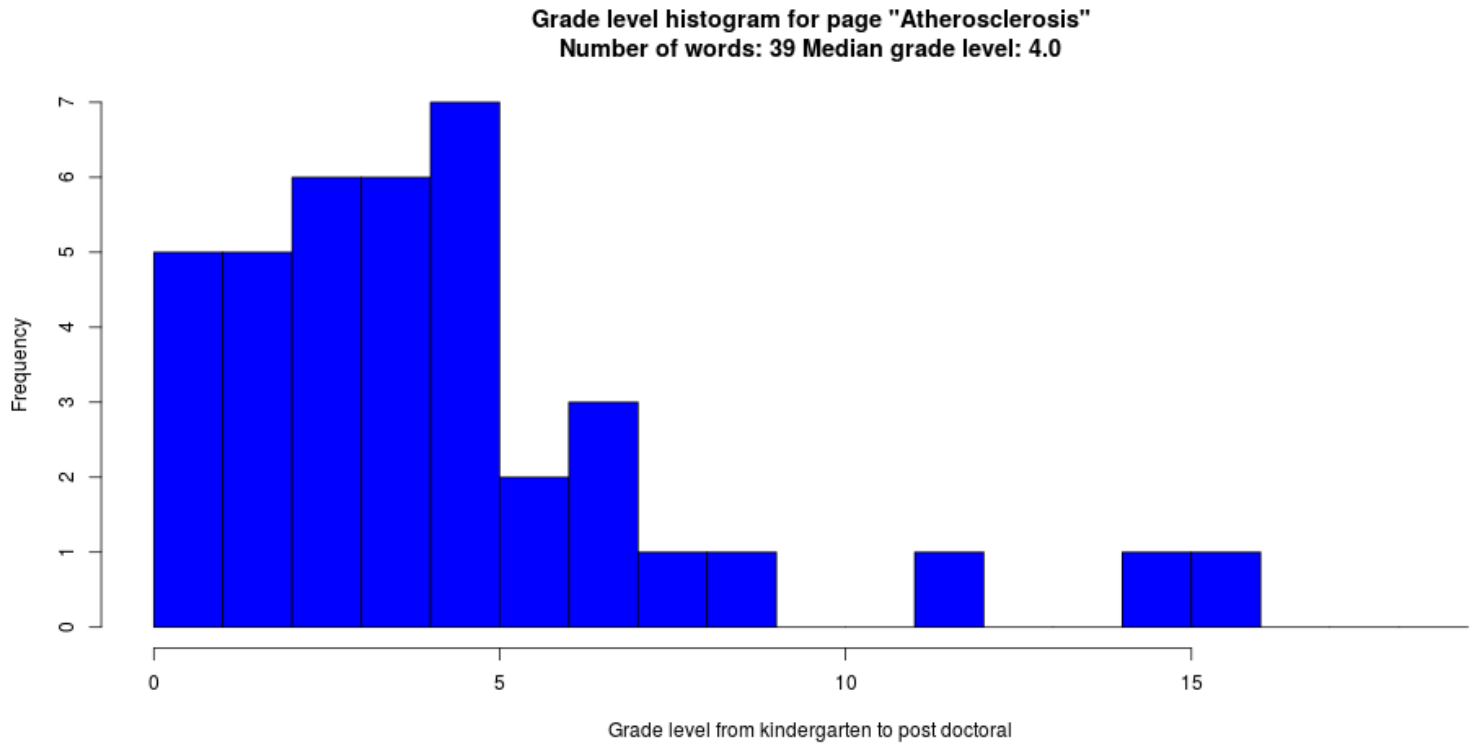

Figure 17: Page Atherosclerosis histogram of grade level vocabulary.

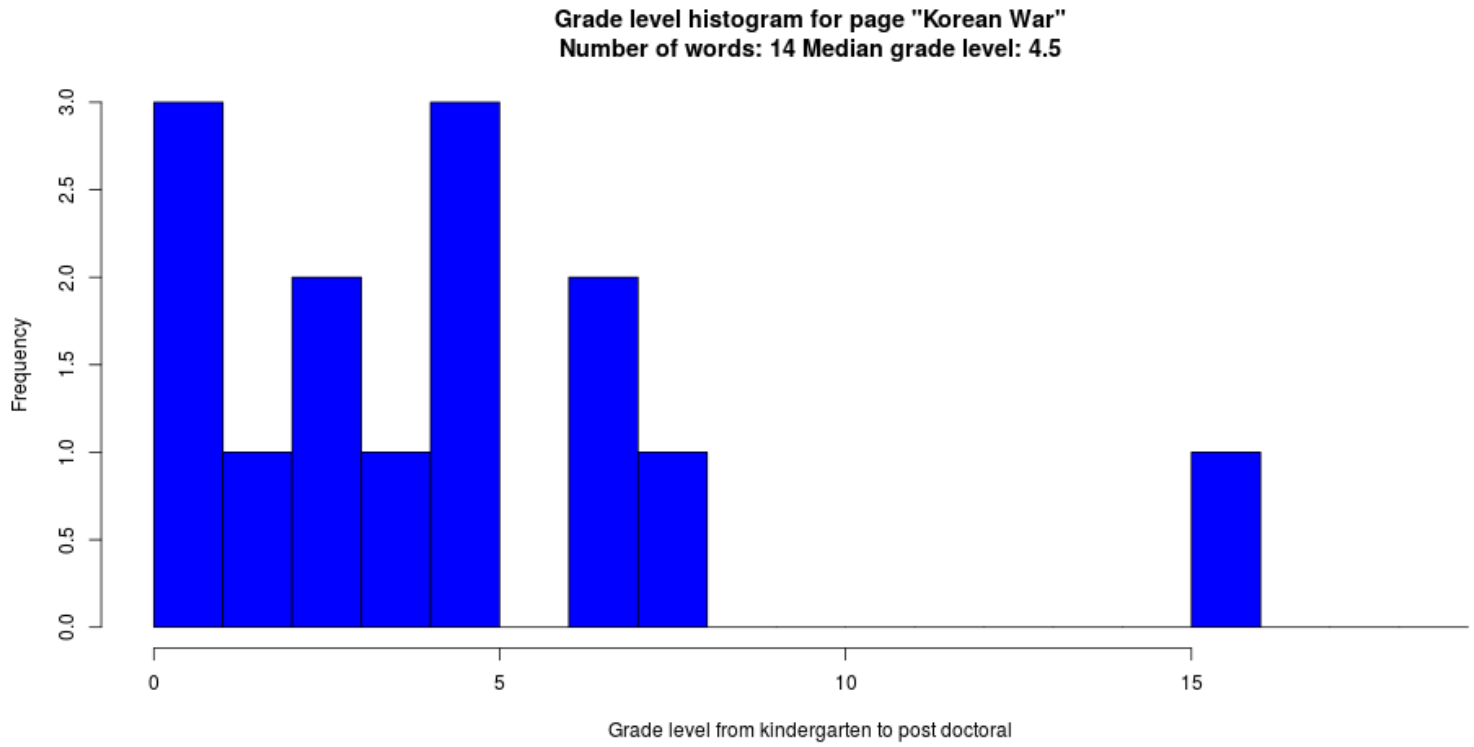

Figure 18: Page Korean War histogram of grade level vocabulary.

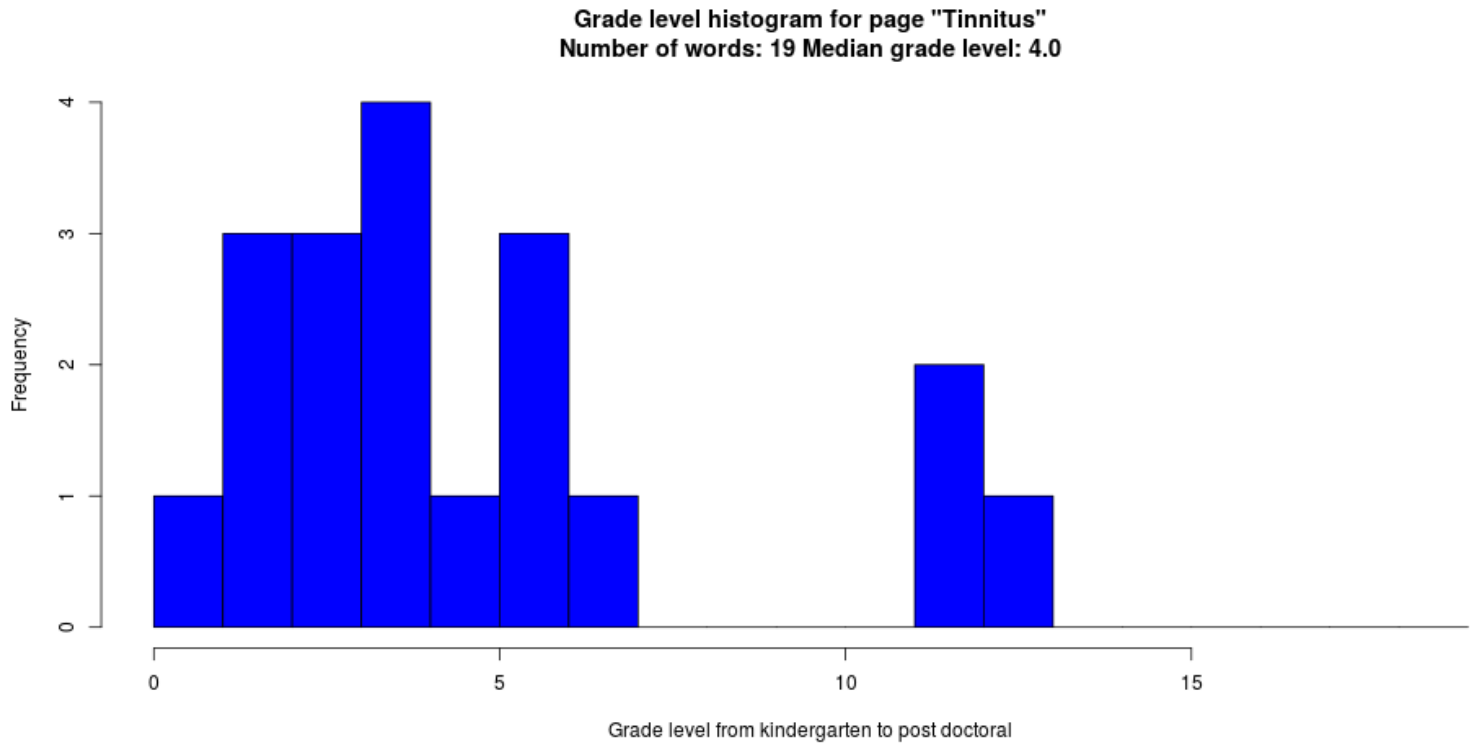

Figure 19: Page Tinnitus histogram of grade level vocabulary.

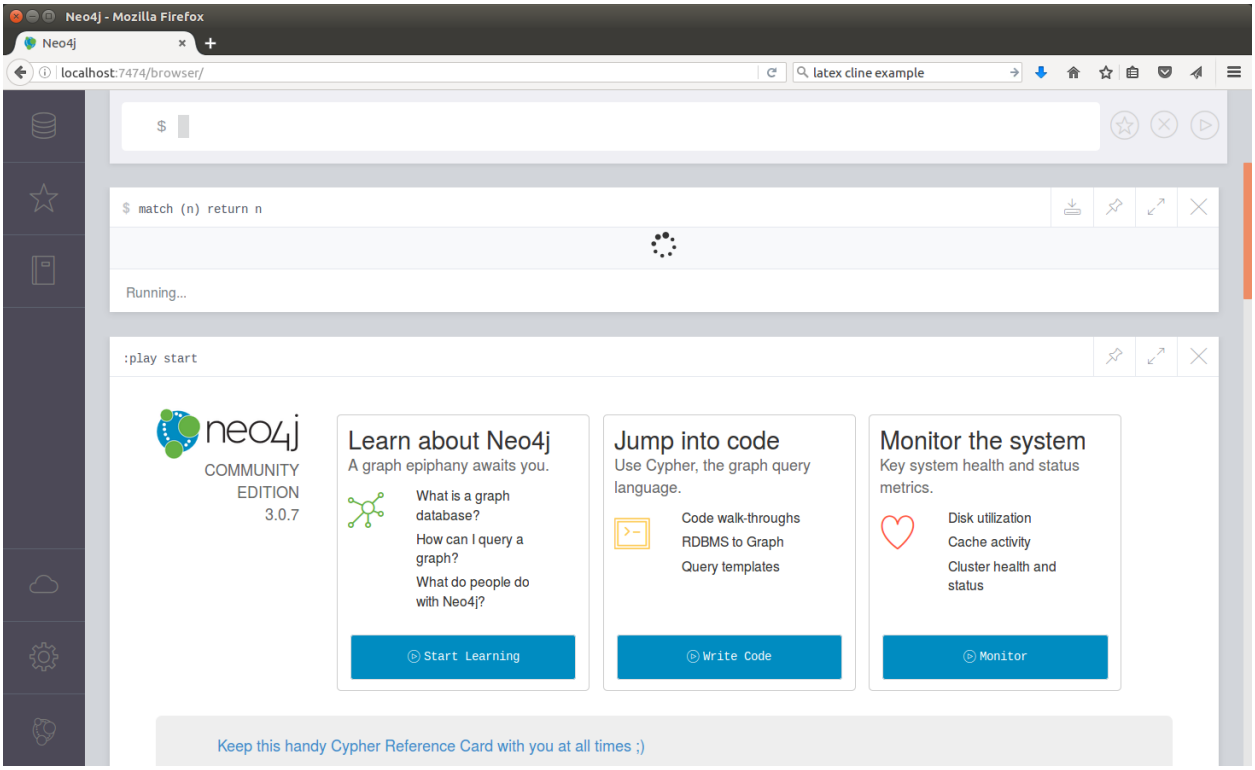

Figure 20: Attempting to view Neo4j graph in browser. `http://localhost:7474/browser/`

4 Conclusion

Don't do this at home. Or anywhere else.

We didn't achieve all that we wanted. The original goal was to be able to connect any two arbitrary Wikipedia pages and look at the educational level of the pages that were connected from the source to the destination pages. Based on the lessons learned we can say this:

1. Wikipedia makes their database available for download, this might be a very good alternative to individual page querying.
2. The number of outgoing links from a page can vary from a few tens to a few thousand.
3. Based on the small sample downloaded from Wikipedia, it would take about 2 months of clock time to down load Wikipedia using 9 crawlers.
4. The number of simultaneous crawlers is limited by the amount of available RAM, because each crawler takes about 70K bytes when running.
5. A PostgreSQL database can be an adequate process scheduler.
6. A Neo4j database can answer the question of how arbitrary pages are connected.

Based on the sample pages processed, the median grade level is about 5. It was fun to develop the R and psql scripts to explore Wikipedia.

5 References

[1] Douglas Adams, *Dirk gently's holistic detective agency*, Simon and Schuster, 1991.

[2] BUSD Staff, *Busd grade level academic vocabulary*, `http://www.berkeleyschools.net/wp-content/uploads/2013/05/BUSD_Academic_Vocabulary.pdf`, 2016.

A Misc. files

The files used to create all these figures are attached to this report. They are:

1. library.R – a collection of functions used by other scripts
2. nodes.psql – PostgreSQL select command to get a list of unique pages/nodes
3. relations.psql – PostgreSQL select command to get a list of connections/arcs
4. wikipediaExplore02.R – an R script to explore the Neo4j graph
5. wikipedia.R – an R script to download Wikipedia pages and update the PostgreSQL database
6. wikipediaStats.R – an R script to analyze the performance of the system and create plots
7. wikipediaStats.R – an R script to analyze the performance of the system and create plots
8. Makefile – a makefile to coordinate and manage all exploration activities stopwords.txt – a list of common English stop words