Graph Visualization Using Unstructured Text

Brendan Hawk

2021-10-19

Bigram Graph Visualization

Data Acquisition and Pre-Processing

For this task, I've chosen to retrieve open text data in the form of a public domain publication from Project Gutenberg[1]. At the time of writing we are nearing Halloween, so I've made the timely choice of selecting **The Dunwich Horror** by H. P. Lovecraft.

Accessing data from Project Gutenberg is very easy on their website, as well as programatically thanks to the `gutenbergr` package for `R`. The file comes in a text format, with each line of the book (more or less as you'd find it in a trade paperback publishing) as a separate row in a table.

A Note about downloading data using **`gutenbergr`*****:*** *The list of available mirrors for Gutenberg downloads can be a bit unstable. If you receive timeout errors running the code below, try picking a mirror manually from the list available at https://www.gutenberg.org/MIRRORS.ALL and supplying it as an argument to the download function as* `gutenberg_download(mirror = "...")`

```r
# Not Run
tdh <- gutenberg_metadata %>%
  filter(title == "The Dunwich Horror") %>%
  pull(gutenberg_id) %>%
  first() %>%
  gutenberg_download()

save(tdh, file="the_dunwich_horror.Robj")
```

[1]https://www.gutenberg.org/

Table 1: Sample of *The Dunwich Horror* as Tibble

gutenberg_id	text
50133	*The* Dunwich Horror
50133	
50133	by H. P. LOVECRAFT
50133	
50133	
50133	"Gorgons, and Hydras, and Chimeras–dire stories of Celæno and
50133	the Harpies–may reproduce themselves in the brain of
50133	superstition–*but they were there before.* They are transcripts,
50133	types–the archetypes are in us, and eternal. How else should the
50133	recital of that which we know in a waking sense to be false come

Creating Tidy Bigrams

There is *much* pre-processing that could be done to this book. Owing to the age of the book, as well as the author's penchant for being *creative* with the English language, there are a great many non-standard words in the text. Processing for this analysis has many steps:

1. The entire book is re-encoded from `LATIN-1` to `UTF-8`. Not every book from the Project Gutenberg library comes in the same encoding, so some exploration will almost always be required to know what you're dealing with.
2. Underscores removed - in this case it is used in the same way as markdown, to italicize words.
3. There is copious use of the special *æ* character that I have chosen to replace with standard letters as "ae".
4. Lovecraft famously uses colloquial accents in his dialogue, and while I could spend hours creating a complete dictionary for normalization, I have chosen to only fix the most egregious case here - He often exaggerates words such as "about" and "house" by spelling them as "abaout" and "haouse". I have removed the extraneous "a" in such cases.
5. "'s" at the ends of words are removed. I was afraid here that lemmatization or stemming tools would remove too many tokens outright (see the point above) so I wanted to do this more manually. Again, I could do considerable more work to make this totally normalized, but won't for this analysis.
6. Tokens that are only numbers are removed. There is extremely few of them in the bigrams, and they don't add much to the analysis.
7. Per the instructions of the assignment, I've removed the words "The", "Dunwich", and "Horror". This is probably not actually beneficial given this text source, but if the analysis relied on them it would be trivial to put them back into play for future work.

At this point, one final manual step is performed before bigrams are tokenized. Due to the way this text source is formatted - again, in the style of a trade paperback - there are several lines of text that are the final word in a sentence. If I ran these as-is through a tokenizer specifically configured to look for bigrams, these lines would return `NA` values and these words would be lost. This is unnecessary selection, so I will leverage `dplyr`'s window functions (ie `lag` or `lead`) to append these words to the lines before them. There is no need to remove them - they will be removed by the tokenizer and therefore will not be duplicated in final counts.

Lastly, the bigrams are parsed out of the text, and aggregated (this will happen twice!) so that we can begin to get the intended edge weight values. Bigrams that contain stop words are also removed at this point. The bigrams are finally split into individual tokens in separate columns of the table we're working with so that they can become the vertexes in our graphs later.

At this point, it is time to construct the undirected network. The weights are only partially aggregated by now - for example if a bigram "token1 token2" appears as well as its reciprocal "token2 token1", these still need to be aggregated in order to get the final edge weight between those two vertexes. Instead of doing complicated table joining, I'll rely on the `igraph` function `simplify` with a parameter for combining the edge weights (`edge.attr.comb = "sum"`) and it can do all the work for me.

Visualizing Bigrams from The Dunwich Horror

Now, with the network created, we will visualize it. This network is **very large**, with 2314 vertexes and 2182 edges. We can take a peek at just the edges with the highest weight - this gives us the bigrams with most frequently co-occurring tokens.

Table 2: 10 Most Co-occuring Tokens in *The Dunwich Horror*

token_1	token_2	weight
armitage	dr	15
sothoth	yog	15
hill	sentinel	13
whateley	wilbur	13
glen	spring	9
bishop	seth	7
corey	mis	7
earl	sawyer	7
hill	noises	7
altar	stone	6

Now, for illustrative purposes, let's take a look at the entire network. Here I have only labeled the edge weights if they are greater than 1, and nothing else.

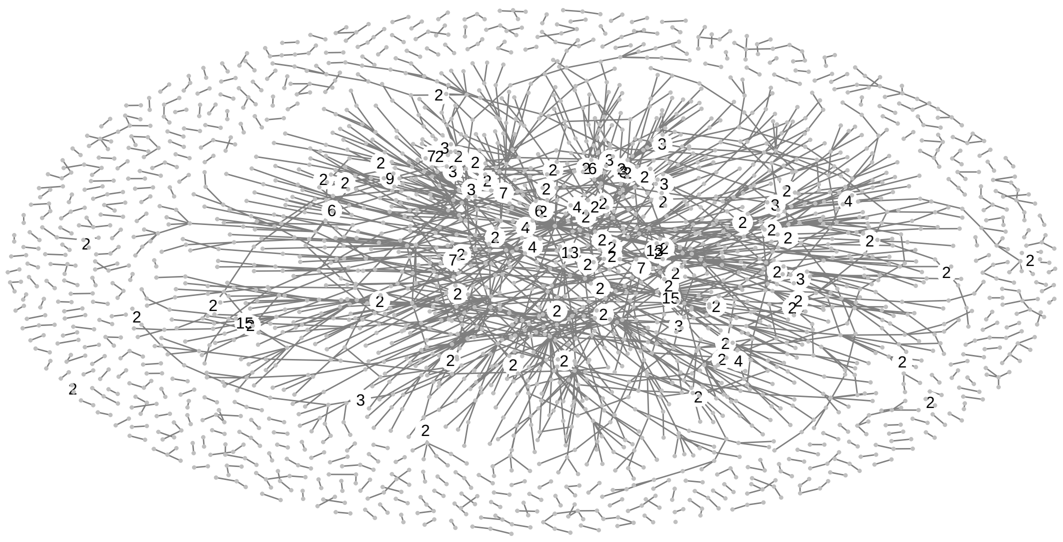

Figure 1: Bigram Network for **The Dunwich Horror**

This is obviously completely illegible. There are simply too many edges and vertexes. We can begin to cut this back to subgraphs of some meaningful importance. This will aid us in distinguishing more important vertexes.

For a first subgraph I have drawn below the subgraph made of only vertexes with a Betweenness Centrality greater than 0.01. This is a somewhat arbitrary value. In this case, it was chosen because for this network it is the beginning of the long tail of the histogram of Betweenness scores. For this and all charts below, the vertexes will now be sized by their total (weighted) degree. Filtering by Betweenness would normally show vertexes that are likely to appear in the middle of reasonable paths, but because our network is undirected we have allowed the order of the words to be somewhat jumbled, so this may not be all that informative.

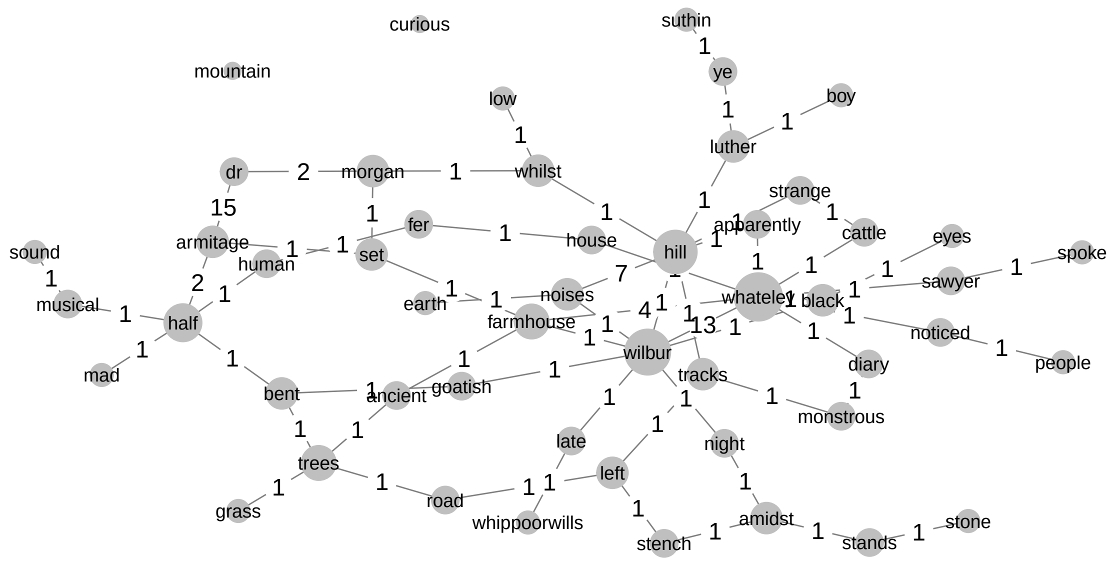

Figure 2: Subgraph of Bigram Network for **The Dunwich Horror**, Vertexes with Betweenness > 0.01

We can even further scale a subgraph back to get at the most important edges. Below is a graph of only those edges with weight greater than 1. The vast majority of edges are of weight 1 so scaling the edges to only weights above this drastically reduces the noise and makes a much more readable graph.

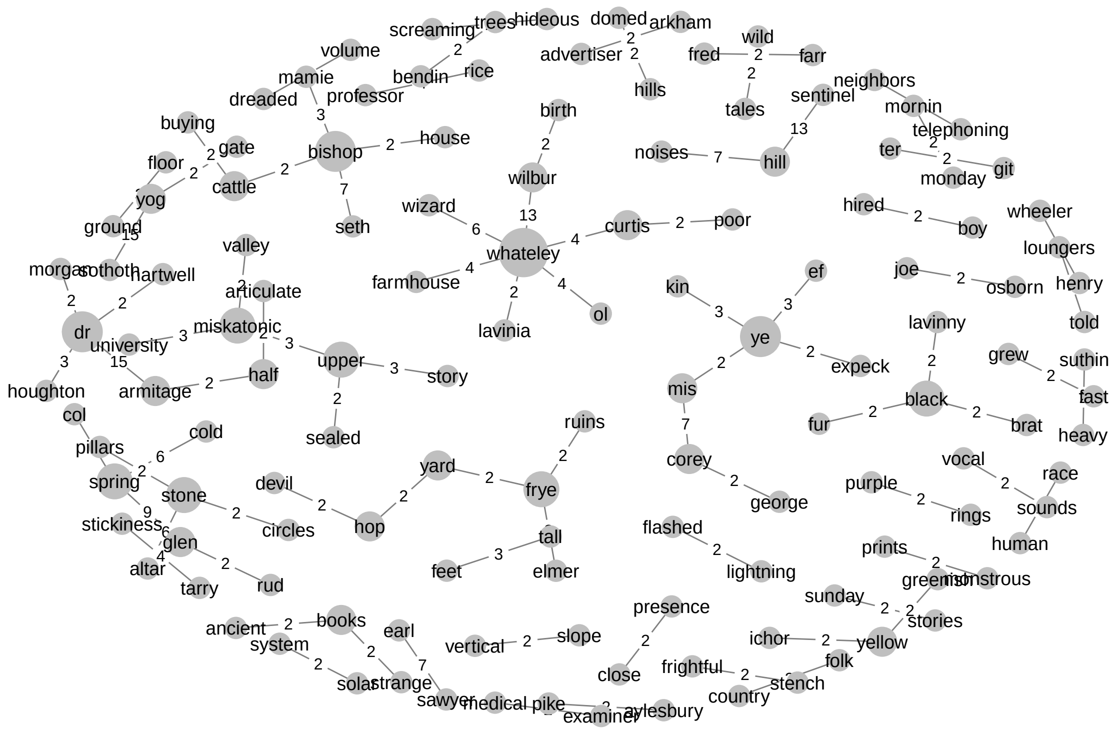

Figure 3: Subgraph of Bigram Network for **The Dunwich Horror**, Edges with Weight > 2

This last graph begins to demonstrate the words that are the most commonly co-occuring. There are some obvious patterns here: names that are two parts such as “Professor Rice” appear to have a heavily weighted connection. The chant “Yog Sothoth”, spoken by cult members in the story, is chanted repeatedly and therefore has a high weight. It also shows some interesting spoke-and-hub shapes, such as the name Whateley being surround by many words that preceed it when used to refer to a person (“Wilbur”, “Curtis”, “Ol”’, etc) and one that follows it to name a place “Whateley Farmhouse”. We can also see in the middle some of the colourful colloquial language we didn’t clean up in pre-processing, such as “Kin ye” (can you), “Ye expeck” (you expect), etc. This and a few other sets like it (“Miskatonic University”, and “Upper Miskatonic”, as more examples) show how the undirected nature of this graph take out some of the meaningful linguistic information. With perhaps more “paper” space and more of his works in this form, we could begin to construct an entire “grammar” of Lovecraft’s colourful and creative use of the English language.

Appendix A: Code

```r
# Import required libraries, installing any that are missing
required_libraries <- c(
  "dplyr",
  "igraph",
  "ggplot2",
  "RColorBrewer",
  # an extension to ggplot2 that provides ggnet2, a ggplot2-based
  # plotting function for network graphs
  "GGally",
  # a required package for GGally that it will not install itself
  "sna",
  # Allows GGally functions to use igraph objects
  "intergraph",
  "gutenbergr",
  "tidytext",
  "tidyr"

)

for (library_name in required_libraries) {
  if (!require(library_name, character.only = TRUE)) {
    install.packages(library_name, repos = "http://cran.us.r-project.org")
    library(library_name, character.only = TRUE)
  }
}

# By default, keep to a readable amount of decimal places
options(digits = 3)

# For reproducibility, set the seed manually.
set.seed(2021)

# Set a base theme for all ggplot2 charts
theme_set(theme_void())

# Clean up unneeded vars
remove(list = ls())
```

```r
# Not Run
tdh <- gutenberg_metadata %>%
  filter(title == "The Dunwich Horror") %>%
  pull(gutenberg_id) %>%
  first() %>%
  gutenberg_download()

save(tdh, file="the_dunwich_horror.Robj")

load("the_dunwich_horror.Robj")

tdh %>%
  mutate(text = iconv(text, "LATIN1", "UTF8")) %>%
  head(10) %>%
  knitr::kable(caption = "Sample of *The Dunwich Horror* as Tibble")

tdh_clean <- tdh %>%
  # we don't need the first column
  select(-gutenberg_id) %>%
  # remove first 5 lines (front matter)
  slice(6:nrow(tdh)) %>%
  # re-encode as utf-8
  mutate(text = iconv(text, "LATIN1", "UTF-8")) %>%
  # remove formatting characters (used for italics, a la markdown)
  mutate(text = gsub("[_]", " ", text)) %>%
  # replace æ character with "ae"
  mutate(text = gsub("\xe6", "ae", text)) %>%
  # Lovecraft often uses "ao" to emphasize spoken language idiosyncrasies,
  # such as "abaout" instead of about, or "maouth" instead of "mouth"
  # normalize this for this text.
  # There are a LOT more of this kind of stylization, but for brevity this is the
  # only one I will fix here. In another analysis I might create a lookup dictionary
  # to normalize all of his colloquial dialogue.
  mutate(text = gsub("ao([uw])", "o\\1", text)) %>%
  # I could write dozens of more manual steps for this, but the last one I'll do
  # is removing `'s` at the end of words
  mutate(text = gsub("'s\\b", "", text)) %>%
  # Remove completely blank lines
  filter(text != "") %>%
  # remove lines that are just chapter numbers
  filter(!grepl("^[[:digit:]]+$", text)) %>%
  # per assignment instructions, remove "the dunwich horror" from our text
  mutate(text = gsub("\\b(the|dunwich|horror)\\b", " ", text, ignore.case = TRUE))

# There are a few lines of text here that are only one token long.
# I'd like to append them to the line before them so that we can capture
# these as part of the bigrams when we tokenize it later.
# This is accomplished using dplyr window functions (lead/lag)
tdh_clean <- tdh_clean %>%
  mutate(
    # IF (a next line of text exists, and it does not contain a space)
    # THEN (append the next line of text to the current one)
    # ELSE (leave it alone)
    text = ifelse(
      !is.na(lead(text)) & !grepl(" ", lead(text)),
      paste(text, lead(text)),
      text
    )
  )
# There is no need to remove the one-token lines, they'll be stripped
# automatically when we parse into bigrams

# tokenize into bigrams, removing any NA's
tdh_bigrams <- tdh_clean %>%
  unnest_ngrams("bigram", "text", n = 2) %>%
  filter(!is.na(bigram)) %>%
  # group by bigram and get frequencies
  group_by(bigram) %>%
  summarize(n = n()) %>%
  # split bigram into tokens (vertexes in our networks later)
  separate(bigram, c("token_1", "token_2"), sep = " ", remove = FALSE) %>%
  # remove bigrams if EITHER token is a stop word
  filter(!(token_1 %in% stop_words$word | token_2 %in% stop_words$word))

tdh_g <- tdh_bigrams %>%
  # select the tokens as our vertexes and use our counts as edge weights
  select(token_1, token_2, weight = n) %>%
  graph_from_data_frame(directed = FALSE) %>%
  # this will combine multiplex edges,
  # ie "token1 token2" and "token2 token1" will be combined into a single edge
  # and it's weights will be summed for a final edge weight
  simplify(edge.attr.comb = "sum")

# Get all the edges of our network, with their weights
edges <- E(tdh_g)
# order them by their weights
edges_ordered <- edges[order(edges$weight, decreasing = TRUE)]
# Grab the tokens from the 10 highest weighted edges
tibble(bigram = as_ids(edges_ordered[1:10])) %>%
  separate(bigram, c("token_1", "token_2"), sep = "\\|") %>%
  mutate(weight = edges_ordered[1:10]$weight) %>%
  # arrange in order of descending weight
  arrange(desc(weight)) %>%
  knitr::kable(caption = "10 Most Co-occuring Tokens in *The Dunwich Horror*")

# Visualize the entire network with minimal labelling
tdh_g %>%
  ggnet2(
    size = 0.1,
    edge.label.size = 2,
    edge.label = ifelse(edges$weight > 1, edges$weight, ""),
    edge.label.fill = ifelse(edges$weight > 1, "white", NA)
  ) + theme(legend.position = "none")

# Trim the network to only vertexes with betweenness > 0.01, and visualize
# the subgraph
tdh_g %>%
  induced_subgraph(V(tdh_g)[igraph::betweenness(tdh_g, normalized = TRUE) > 0.01]) %>%
  ggnet2(
    size = "degree",
    max_size = 8,
    label = TRUE,
    label.size = 2.5,
    edge.label = "weight",
    edge.label.size = 3,
  ) + theme(legend.position = "none")

# Trim the subgraph to only edges with weight > 1, and visualize
tdh_g %>%
  subgraph.edges(edges[edges$weight > 1]) %>%
  ggnet2(
    size = "degree",
    max_size = 8,
    label = TRUE,
    label.size = 2.5,
    edge.label = "weight",
    edge.label.size = 2,
    mode = "kamadakawai"
  ) + theme(legend.position = "none")
```