Path to Love

Fabricio Mougou
fabricio.mougou@durham.ac.uk

Christmas 2024

To Hazel,
It took us years to find love.
Our words have ensured it forever lingers,
only a few steps away.

1 Introduction

Art is inherently human, almost as much as the use of tools is. Therefore, it is no surprise that humans have always turned tools into instruments for expression, from the brush, to the pen, to even the bucket. I believe that among all, the computer is the most effective tool, its capabilities offer precision, scale, and depth to a digital canvas. By harnessing the computational power of modern systems, I aim to craft a piece of art that maps the network of words shared between me and Hazel, and doing so with a focus on love.

2 Methodology

To create 'Path to Love', I had to go through three phases: Data collection, data processing, and finally data visualization. All of the project was undertook using python.

2.1 Data collection

I collected all the messages sent between me and Hazel on iMessages. This data could be found on my system. Using SQL, I queried the database to extract the messages sent between us, but due to a common bug, the 'text' fields of a significant number of messages were 'NULL'.

However, the text values were hidden in the binary contents of the file [1], and I was able to extract the messages using the "imessage-tools" library for python [2], with some minor changes to also gather if a message was sent from me or not. Interestingly, 17491 messages were gathered, meaning approximately 24 messages were shared between us every day since we met.

2.2 Data processing

To obtain the unique words used in our messages, it took a few steps. I filtered through the messages, split them into arrays using whitespace as a delimiter. Then, for each word in each message, I removed them from the array if they were empty or just whitespace, and removed all characters that were not alphanumeric. Finally, I added these words to a dictionary - the key being the word and the value initially being 1 and incrementing whenever a repeat was encountered. After these steps, the total number of unique words was 6357.

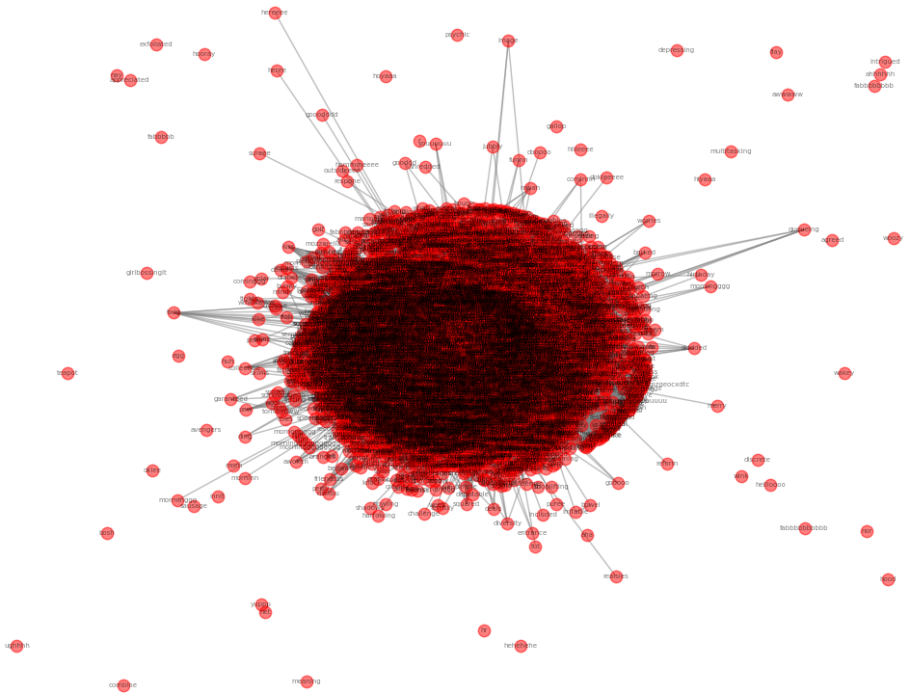

Figure 1: Co-occurrence network using 6357 words

As shown in Figure 1, this was too many words to work with, the number of nodes and edges in the co-occurrence graph (following the not yet mentioned data processing steps) created a plot that was too cluttered for my vision. To address this issue, I applied extra filtering of words by checking each word against the English dictionary using the pyEnchant library [3].

If a word was not in the English dictionary, then I checked if they occurred more than 5 times (acting as a threshold for slang) and removed them if they did not. This brought the total number of unique words down to 4571.

Figure 2: Word filtering from a message

A co-occurrence matrix will allow me to capture relationships between different words. With the unique words labelling both the rows and columns, and each entry being filled with 0 initially. To fill it, if two words appear in the same sentence I increment the entry at (word1, word2).

$$\text{Let } M = \{m_1, m_2, ..., m_n\} \text{ be a set of } n \text{ messages}$$
$$\text{where } m_i = \{w_1, w_2, ...w_p\} \text{ is a list of } p \text{ words.}$$

$$\text{Let } W = \{w_1, w_2, ..., w_k\} \text{ be a set of all } k \text{ unique words.}$$

$$C \in \mathbb{N}^{n \times n} \text{ be a co-occurrence matrix.}$$

$$C = \begin{array}{c|cccc} & w_1 & w_2 & \cdots & w_k \\ \hline w_1 & 0 & 0 & \cdots & 0 \\ w_2 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_k & 0 & 0 & \cdots & 0 \end{array}$$

$$\text{Let } C_{ij} = \sum_{m \in M} \delta(w_i, w_j, m)$$

$$\text{where } \delta(w_i, w_j, m) = \begin{cases} 1 & \text{if } w_i \in m \wedge w_j \in m \\ 0 & \text{otherwise} \end{cases}$$

Note that the matrix is symmteric, meaning that $C_{ij} = C_{ji}$.

With the co-occurrence matrix, the data is processed to the point of being able to be visualised. When plotting the co-occurrence network $G = (W, E)$, we know that if $C_{ij} > 0$ then $w_i w_j \in E$.

If there is a value between two words in the co-occurrence matrix, an edge is drawn between the two words in the network.

	w_1	w_2	w_3	w_4	w_5
w_1	0	1	0	2	0
w_2	1	0	3	0	1
w_3	0	3	0	4	0
w_4	2	0	4	0	5
w_5	0	1	0	5	0

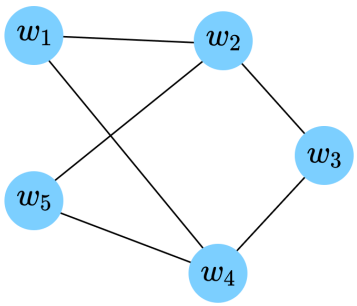

Figure 3: Example co-occurrence matrix & network

The network in Figure 1 shows the co-occurrence network of 6357 unique words in our messages.

Using our matrix, we can see the paths between two words by looking if there are a sequence of edges we can traverse to get from one word to the other.

	w_1	w_2	w_3	w_4	w_5
w_1	0	1	0	2	0
w_2	1	0	3	0	1
w_3	0	3	0	4	0
w_4	2	0	4	0	5
w_5	0	1	0	5	0

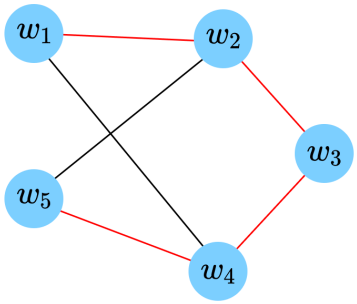

Figure 4: Path from w_1 to w_2

The basis of my art will be on plotting the shortest path between the word 'love' and all other words. To do so, I initialised a subgraph of G, $G_s = (W, E_s)$, where E_s only contains the shortest paths from $w_i \in W$ and our target word love, w_{target}. I leveraged Dijkstra's algorithm [4] to create this new set of edges.

$$E_s = \{w_i w_j \in E : d(w_i, w_{target}) - d(w_i, w_j) = d(w_j, w_{target})\}$$

Where $d(w_x, w_y)$ uses Dijkstra's and returns the shortest path between words $w_x, w_y \in W$.

2.3 Data Visualisation

Initially, I considered using the spring layout [5] for visualising G_s. This creates a force-directed graphs, meaning that nodes in the graph have attractive and repellent tendencies between each other.

Figure 5: G_s plotted using spring layout

2.3.1 Spring Layout

Spring layout applies an attractive force between connected pairs of nodes, and a repelling force between all nodes in the graph. Initially, all nodes are given a random position in the graph. For my usage, for all co-occuring words $w_i, wj \in W, w_i w_j \in E_s$ the attractive force is

$$F_a(w_i, w_j) = k_a ||pos(w_i) - pos(w_j)||$$

where: k_a is a constant of attraction, controlling the strength of the attractive force, and the repulsive force is defined by

$$F_r(w_i, w_j) = \frac{k_r}{||pos(w_i) - pos(w_j)||}$$

where k_r is a constant of repulsion, controlling the strength of the repulsive force. The values for k_a and k_r used are $\sqrt{\frac{1}{|W|}}$ and $\frac{1}{\sqrt{|W|}}$ respectively. The goal of the algorithm is to minimise the energy of the system, where the energy is defined by

$$F_{energy} = \sum_{w_i w_j \in E_s} F_a(w_i, w_j) + \sum_{w_i, w_j \in W} F_r(w_i, w_j)$$

I was not satisfied with the results plotting G_s with the spring layout produced, as it looked too chaotic and did not highlight 'love' adequately. I also experimented with using the Kamada-Kawai layout [6], but as this is also a force-directed graph layout, it produced similar results.

Figure 6: G_s plotted using Kamada-Kawai layout

2.3.2 Custom Layout

Finally, I decided on using a custom layout that splits up the words into layers depending on their distance in steps to 'love'. This will put the word love at the top of the graph at layer 0, all words directly co-occuring with it at layer 1 below, and so on. To visualise the layout, I attributed a layer to each word, with the layer being the value of the length of the shortest path. Note that the maximum layer was 3, and nodes without any paths to love were removed. Then, I uniformly split the words on each layer horizontally so that they are evenly spaced, and aligned them to the center. Next, I assigned a random y co-ordinate to each word, within the range (-layer + margin_top, -layer - margin_bottom).

$$\text{margin\_top} = 1.8$$

$$\text{margin\_bottom} = \begin{cases} 1 & \text{if } 0 \leq \text{layer} \leq 2 \\ 1.6 & \text{if layer} = 3 \end{cases}$$

Furthermore, I set the node sizes in the graph dynamically, where the size for any word is $size(w_i) = max(5 \cdot degree(w_i), 30)$, where $degree(w_i)$ is the number of edges in which a word appears. To further fine tune the piece I changed the colour scheme of the graph for nodes to have different colours depending on the layer they are in, set the colours of the edges to be a midpoint between the colours of the layers it traverses, and reduced the opacity of both the nodes and the edges to better visualise them. The last step I took was to split up the messages into ones sent by me and ones sent by Hazel, and followed the process through with each set to create separate graphs with fewer words with the objective of making them look less cluttered, combining them later on.

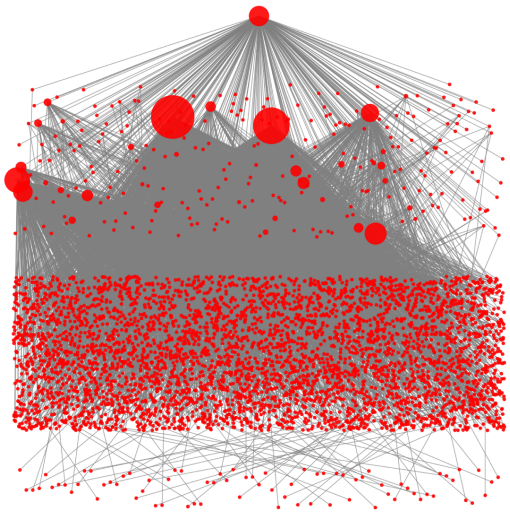

Figure 7: G_s plotted using layered layout

2.4 Conclusion

All that is left to do is to merge the two separate graphs of words spoken by Hazel and I at the node for love. This created an hourglass sort of shape, which I think offers beautiful symbolism about the passing of time in our relationship, of patience and balance, of the fleeting moments, of the steady flow of shared hours, of exploring our pasts, and of the eagerness for the future.

Interestingly, the co-occurrence graph of our combined words makes up a small-world network. Although this may not have any implications for a scholar, its poetic resonance is undeniable. Personally, I see it as a metaphor for how words - in their vast complexity - connect us, and conversations that may have seemed inconsequential at a time may in fact be the ones that lead you to passion, connection, and love.

The project was implemented using Python and several libraries, the source code is available at `https://github.com/FabMougou/path-to-love`.

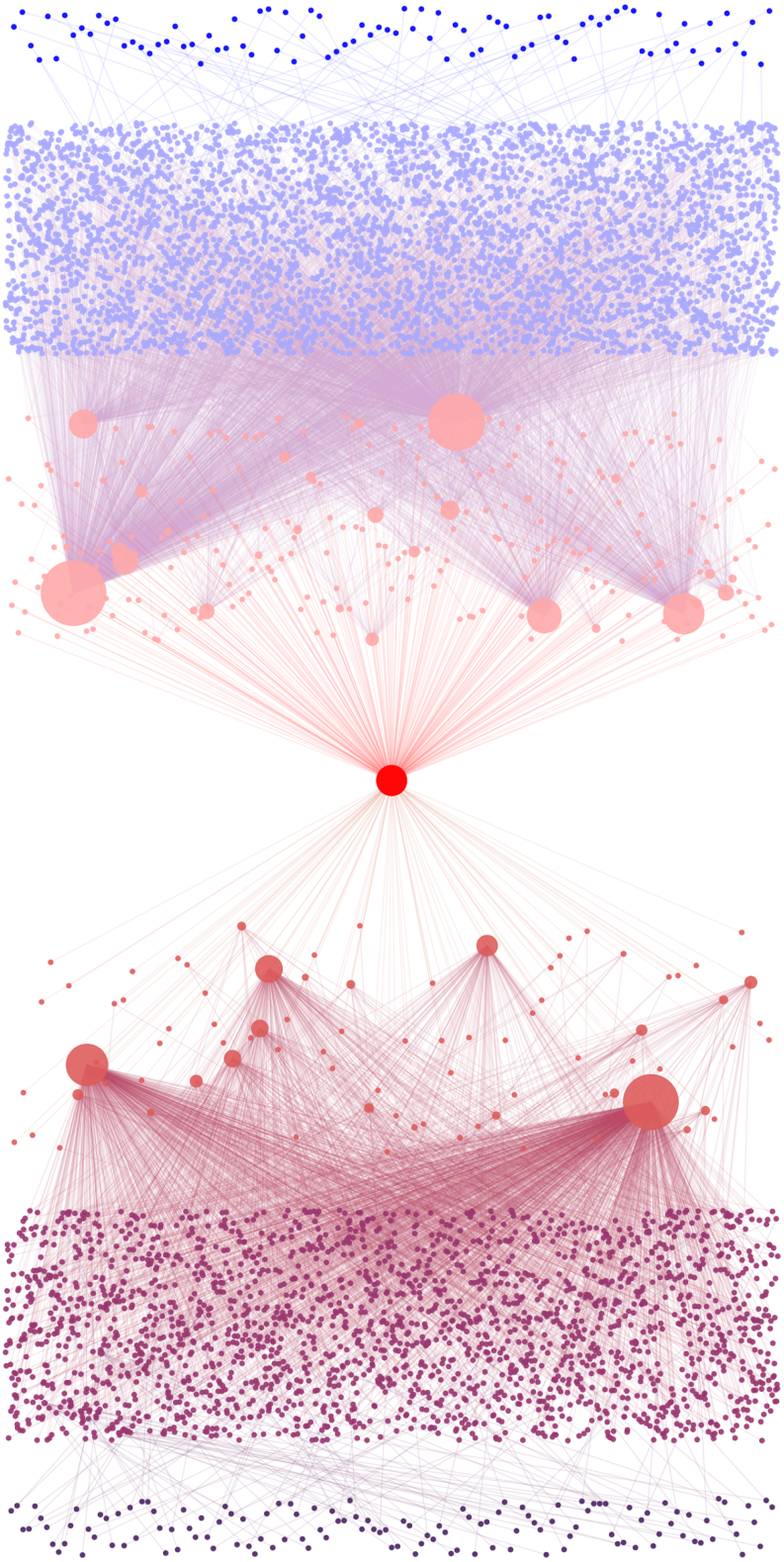

Figure 8: Path to Love

References

[1] u/caleb531. *Comment in: Texts are missing from Mac chat.db file despite showing in iMessage*. Accessed: 2024-12-16. 2022. URL: https://www.reddit.com/r/osx/comments/uevy32/comment/ivfyh0q/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button.

[2] my-other-github-account. *imessage$_t$ools*. Accessed: 2024-12-16. 2023. URL: https://github.com/my-other-github-account/imessage_tools.

[3] Dimitri Merejkowsky. *pyEnchant*. Accessed: 2024-12-16. 2023. URL: https://pypi.org/project/pyenchant/.

[4] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[5] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph Drawing by Force-directed Placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. DOI: 10.1002/spe.4380211102.

[6] T. Kamada and S. Kawai. "An Algorithm for Drawing General Undirected Graphs". In: *Information Processing Letters* 31.1 (1989), pp. 7–15. DOI: 10.1016/0020-0190(89)90102-6.