The phonological component of the Dutch mental lexicon is small-world

Casper van Velzen
BA Taalwetenschap
Universiteit van Amsterdam
26-07-2018

Abstract
Phonological forms of words are stored in the mental lexicon. Previous studies have found that these entries in the lexicon are somehow connected. Using a branch of mathematics called *Graph Theory*, this study analyses this network of phonological forms stored in the brain. The network turns out to be *small-world*, meaning there is a high degree of clustering.

The phonological component of the Dutch mental lexicon is small-world

Casper van Velzen

Intdroduction

It is not controversial to claim that linguistic elements like phonology and semantics are stored somewhere in the brain. In lingustics, the network of linguistic elements that are stored in the brain is called the *mental lexicon*. Evidence for connections between individual elements has been found in lexical-decision tasks. In these experiments, certain prime words can increase reaction time on a target word if they are phonologically related (Lukatela & Turvey, 1990) This inhibition is proof that the mental lexicon is indeed a network.

Little is known about the shape of individual entries in the mental lexicon, let alone the entire network. Luckily, there is a branch of mathematics called *Graph theory* that concerns networks and network analysis. Using graph theory, the present study will test a model of the Dutch mental lexicon for a property called *small-worldness*.

Jeronimus et al. (2017) define small-world networks as being *sparse graphs* with a high *Clustering Coefficient (CC)* and a low *Average Minimum Path Length (APL)*.

A *graph* is essentially what mathematicians call a network, where *points* (also called *nodes* or *vertices*) are connected by *edges*.
Edges can be *directed* or *undirected*. In undirected networks, an edge going from node A to node B is per definition also an edge going from node B to node A. If the graph is directed, a connection from A to B does not always mean a connection from B to A. An example of an undirected graph would be a social network, where every person is represented by a vertex and their connection is represented with an edge. If person A is acquainted with person B, person B is also necessarily acquainted with person A. A directed graph can be found in the simulation of an ecosystem. If a certain species of bird likes to eat worms, the worms do not necessarily eat those birds, so an edge would only go from the bird to the worm, and not the other way around.
Edges can also have a *weight*. The weight is a number representing the distance between two vertices. For example, in a network representing the European railway system, the three hour ride from Amsterdam to Paris will have a higher weight than the one hour ride from Paris to the French city of Amiens. In an *unweighted* graph, all edges have a weight of 1.

One of the conditions for small-world networks is that the network must be sparse. *Sparse graphs* are graphs that have fewer than the maximum number of connections (Barabási, 2015). If the graph is not sparse, it is *dense.* Figure 1 shows a network where every node is connected to every other node. In other words, the graph has the maximum number of connections and is therefore dense. The number of edges in an undirected dense graph can be found using the formula $edges = \frac{n(n-1)}{2}$ where n is the number of vertices in the network. Figure 2 displays a sparse network.

The *density* of a graph can be calculated by dividing the number of edges by the maximum possible number of edges. For example, the density d of the network in figure 1 is $d = \frac{10}{10} = 1$ while the density of the network in figure 2 is $d = \frac{7}{10} = 0{,}7$.

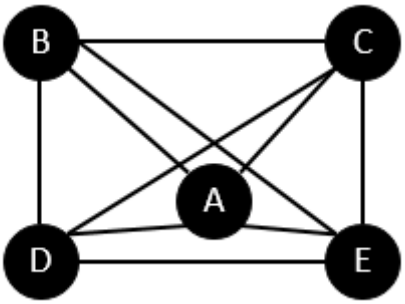

Fig 1: A dense graph

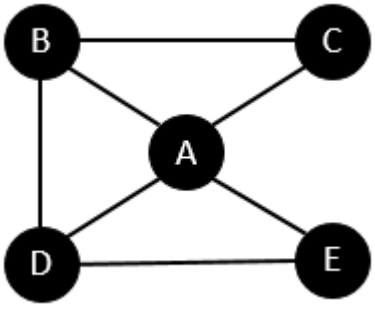

Fig 2: A sparse graph with a CC of approximately 0,767

Another condition for small-worldness is that the graph must have a high *Clustering Coefficient (CC)*. The Clustering Coefficient is a number representing the degree of clustering within a network. The CC of an entire graph is defined as being the average CC of all vertices within the graph.
The CC of a vertex can be calculated as follows: $CC_{vertex} = \frac{nr\ of\ connections\ between\ neighbours}{max\ nr\ of\ connections\ between\ neighbours}$ where the neighbours of a vertex are all nodes that are connected with an edge. In figure 2, the neighbours of node A are the nodes B, C, D and E. The maximum number of edges between these four nodes is 6. Therefore, $CC_A = \frac{3}{6} = 0{,}5$. Similarly, the neighbours of B are A, C and D. Applying the same formula yields $CC_B = \frac{2}{3} \approx 0{,}667$. Taking the average CC of every node in this network yields a result of approximately 0,767.

The final condition for small-worldness is a low *Average Minimum Path Length (APL)*. The minimum path length from one node to another is the shortest route between the two points. Take figure 3. To get from A to B, one could first visit C. In that case, the path length will be 2, because two edges have been traveled. The shortest path however, is the direct connection between A and B, so the minimum path length is 1. Similarly, to get from C to E, one could visit every other node by taking the path C -> A -> B -> D -> E. One could even infinitely loop around. The shortest path, however is C -> A -> D -> E[1] with a path length of 3.
The APL of an entire graph is the average minimum path length from every node to every other node. The minimum path lengths between every vertex in figure 3 can be found in the table of figure 4.

In network research, it is possible to find a network like in figure 5, where there is no way to get from one point to the other, because there is no connection. In case of figure 5, there is no route that leads to or from node E. If this is the case, the graph is called *disconnected*. In a disconnected graph, it is impossible to calculate the APL. What's more, if there is a node without any connections, like E in figure 5, the calculation for its CC would require a division by 0, which is mathematically impossible. For research' sake, the most common way to deal with this problem is to only analyse the *biggest component* of the graph. That is, the part that is connected with the highest number of nodes.

[1] C -> B -> D -> E is an equally short route, but the minimum path length stays at 3.

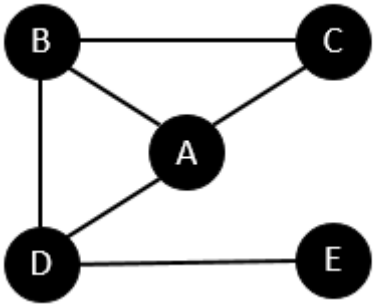

Fig 3: A network with an APL of 1.5

PATH	MIN PATH LENGHT
A -> B	1
A -> C	1
A -> D	1
A -> E	2
B -> C	1
B -> D	1
B -> E	2
C -> D	2
C -> E	3
D -> E	1
AVERAGE	1.5

Fig 4: The minumum path legths and APL of the network in figure 4

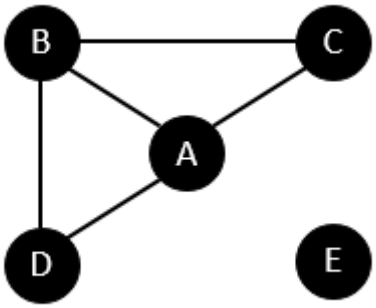

Fig 5: A disconnected graph. The biggest component contains nodes A, B, C and D.

Humphries & Gurney (2008) propose the following formula to quantify small-worldness: $\sigma = \frac{\frac{CC}{CC_r}}{\frac{APL}{APL_r}}$ where the network is small-world if $\sigma > 1$, $CC \gg CC_r$ and $APL \approx APL_r$. The subscript r stands for *randomised*. Determining small-worldness within a network is done by comparing the graph to *Erdos-Rényi (ER) randomised* graphs.

An ER-randomised graph starts with a network of a given size and 0 connections. Then, an algorithm goes over every possible combination of nodes within the network and decides wheter or not to lay a connection, with a given probability. In order to make the randomised graphs and the empirical network comparable, the size and chance of laying a connection should be such that the number of nodes and connections are similar those of to the biggest component of the empirical network. As it happens, all networks I generated were fully connected, so they were very much comparable to the biggest component of the real-world network.

Small-world networks have been found in a number of different fields, ranging from social relations (Wasserman & Faust, 1994) to power grids (Watts & Strogatz, 1998). In linguistics, small-worldness has been found in networks concerning East-Asian orthography (Jeronimus et al., 2017), lexical co-occurrence (Ferrer & Solé, 2001), semantic relations of words (Motter et al., 2002) and the phonology of English lexemes (Vitevitch, 2008).

This study focuses on the phonological component of the mental lexicon. Specifically, how the individual entries are connected on a phonological level. While network analysis cannot give us a clear answer about the exact properties of the mental lexicon, it can give us a clearer view on the way it is structured. It is a puzzle piece that will ultimately help us understand how words are stored in the brain.

The network was created and analysed with a Python script written by me using the Networkx module (Hagberg et al., 2008). The ER-randomised networks were generated using a different self-written script, which relies on the same module. Networkx is useful, as it allows for easy graph creation, and it has readily available functions to calculate Clustering and APL as well as a function to generate ER-randomised graphs. Both scripts are freely available at http://fon.hum.uva.nl/archive/.

Data

All entries in the network were taken from the WebCelex database (http://celex.mpi.nl/, accessed April 2018). The following options were chosen to create the database:

Dutch Lemmas. Lemmas are the conventionalised forms of lexemes that are found in the dictionary. A lexeme is a unit of meaning, without its inflectional variants. Although it can be argued that lexemes are not the only elements explicitly stored in the mental lexicon (Pinker, 1998), it is the bare minimum that has to be stored in the brain.

INL frequency. The Dutch Lemma database has about 125.000 entries, while the average well-educated adult mental lexicon has been estimated to contain about 17.000 entries (Goulden et al., 1990). Therefore, only the 20.000 most frequent entries were chosen. Unfortunately, only about half of the entries actually had a phonological transcription, so the actual number of analysed entries is 10.969.

PhonolCLX. There is a difference between what phonological representation is stored in the brain and the sound that a speaker produces. Since I am only interested in the underlying representation that is stored in the brain, the underlying representation was chosen.

After retrieving the database and filtering out the 20.000 entries with the highest frequency, all morpheme borders (#, + and -) were removed from the transcriptions. Then, all instances of diphthongs were replaced with a number (EI became 1, UI became 2 and AU became 3). For example, the word *hij* ('he') was transcribed as *hEI*. After replacing the diphthong, it is transcribed as *h1*. This way, all diphthongs are counted as a single phoneme. All long vowels were also recognised as a single phoneme within the script. As for why I did not opt for a special character for long vowels, that was the result of the order in which I wrote the code. I had already written the code which combines the two characters representing long vowels into a single element, when I decided to replace all instances of diphthongs with single characters. Since it didn't take a lot of computing power, I decided to let the code stay as is.

Within the network, each lemma is represented by a node, and connections were made if two lemmas are *phonological neighbours*. Phonological neighbourhood is defined as follows:

> *If lemma A results in lemma B when* adding, subtracting or substituting *a single phoneme, lemmas A and B are phonological neighbours*.

A phonological neighbour is not the same as a minimal pair, although all minimal pairs are phonological neighbours. Minimal pairs are useful for determining wheter a sound is a phoneme in a language, but they are not useful for determining wheter or not words are phonologically related. Some words that seem phonologically related, like *vuur* ('fire') and *vuurzee* ('conflagration') are omitted from this definition, because it is difficult to computationally express these connections.[2]

The creation of the network, including the decision as to whether or not to make a connection between nodes, was made in a Python script written by me.

The Clustering Coefficient and Average Path Length were calculated using NetworkX' *average_clustering* and *average_shortest_path_length* functions respectively.

In the set of 10.969 analysed lemmas, 17.066 minimal pairs were found. However, because the network is disconnected, the biggest component of 652 lemmas was analysed instead. Of course, the biggest component is quite small compared to the entire graph, and it might not be representative of the entire network.

	NR OF NODES	NR OF EDGES	APL	CC
ENTIRE GRAPH	10.969	17.066	-	-
BIGGEST COMPONENT	652	5841	3,2033775314994393	0,4139803689756391

Fig 6: Results of the analysis of the empirical network

Randomised networks

1000 ER-randomised graphs were created with NetworkX' *fast_gnp_random_graph* function. The parameters given to the function were *n = 652* and $p = \frac{5841}{(\frac{(n*(n-1))}{2})}$ where n is the number nodes and p is the probability of creating a connection. The size is equal to the biggest component found in the empirical network. p is calculated by deviding the number of connections found in the biggest component of the empirical network by the maximum number of connections in a graph of size n.

Once again, the CC and APL were analysed using the *average_clustering* and *average_shortest_path_length* functions respectively. The data was analysed with R (R Core Team, 2017). As expected, while the number of nodes and edges are similar to those of the empirical network, the CC is a lot lower.

N = 1000	NR OF NODES	NR OF EDGES	APL	CC
AVERAGE	652	5843,28	2,568159	0,02752255
SD	0	72,98	0,008171774	0,001036508

Fig 7: Results of the analysis of the ER-randomised networks

[2] Experienced programmers might argue that an expression along the lines of 'string1 contains string2' will do the trick, but that expression will also connect words like *ze* ('she') and *bewezen* ('proven') which are not related.

Small-worldness

In the introduction, two definitions of small-worldness were mentioned. Firstly, Jeronimus et al. state that small-world networks have a high CC and a low APL.

A z-test shows the number of standard deviations an observation is away from the average of a population. It is calculated with $z = \frac{x-\mu}{\sigma}$ where x is the observation, μ is the average of the population and σ is the standard deviation of the population. The z-score for the CC found in the empiric network is $z = \frac{0{,}4139803689756391 - 0{,}02752255}{0{,}001036508} \approx 372{,}846$
Thus, the empirical network does have a relatively high CC.

While the APL of the empricical network is high compared to the ER-randomised networks, it is still close to the absolute minimum, and far from the absolute maximum APL. The minimum possible APL of a graph can be calculated using the formula $l = 2 - d$ where *l* is the minimum APL and d is the density of the graph (Gulyás et al., 2011). In this case, $l = 2 - \frac{5841}{212226} \approx 1{,}972$. Gulyás et al. also mention that ER-randomised networks usually have an APL close to the minimum.
In the same paper, Gulyás et al. propose a formula for the maximum possible APL in a graph.
$l(N,d) = (dN^2 + (-d-2)\,N + 2) * \frac{\sqrt{4dN^2+(-4d-8)N+9}}{3N^2-3N} + \frac{(1-3d)N^3+(6d-6)N^2}{3N^2-3N} + \frac{(-3d-13)N+6}{3N^2-3N}$
where N stands for the number of vertices in the graph and d once again represents density. With this formula, it is possible to calculate the maximum possible APL for the empirical network:
$l\left(652, \frac{5841}{212226}\right) \approx 203{,}407.$

The second definition for small-worldness is the one proposed by Humphries & Gurney. Calculating σ with their formula results in $\sigma = \frac{\frac{0{,}4139803689756391}{0{,}02752255}}{\frac{3{,}2033775314994393}{2{,}568159}} \approx 12{,}059.$
The conditions $\sigma > 1$ undoubtedly holds true. The other two conditions ($CC \gg CC_r$ and $APL \approx APL_r$) are a little harder to define, since CC is influenced by the density of the graph, and APL is influenced by both density and size.
Because CC is so many deviations removed from CC_r, I argue that $CC \gg CC_r$ holds true. As for the final condition, it is difficult to define 'approximately equal'. It could be argued is that $APL \approx APL_r$ as the z-score is $z = \frac{3{,}2033775314994393 - 2{,}568159}{0{,}008171774} \approx 77{,}733$. However, since both APL and APL_r are close to the mathematical minimum, and far from the mathematical maximum, I would once again argue this criterium is met.

Discussion

The results are in line with previous studies, especially those of Vitevitch (2008), who analysed English phonology and Motter et al. (2002), who analysed semantic relations between words. Between these studies, it is now pretty clear that entries in the mental lexicon are somehow connected in a non-random, structured way. However, the models that have so far been used, including mine, are possibly too simplistic in nature.
First, the method of laying connections between entries might be faulty. The Cohort model for language processing (Marslen-Wilson, 1987) states that when hearing the first phoneme of a word, all entries in the mental lexicon beginning with that phoneme are activated. Then, when hearing a second phoneme, all activated entries that do no match are deactivated. After that, the same

happens for the third phoneme et cetera. If this is the case, weighted connections depending on how similar the beginnings of words are may need to be made.
Second, no model has combined semantics and phonology within a single network. While both individual components have been shown to be small-world, there is no mathematical guarantee that the combination will show the same behaviour.

Another unanswered question is that of how this small-world structure came to be. In other words, why is the network organised in the way that it is? One possible answer is that it is simply a result of the way new words are created. New words can be made by combining or modifying existing words. In this case, new words are likely to be phonologically and semantically related to existing ones. This is mere speculation. However, this question warrants further research.

References

Barabási, A. (2015). *Network science*. Cambridge, UK. Cambridge University Press

Ferrer i Cancho, R. & Solé, R. (2001). The small world of human language. London, UK. *The Royal Society*, 268, pp. 2261-2265

Goulden, R., Nation, P., & Read, J. (1990). How large can a receptive vocabulary be?. *Applied Linguistics*, 11, 341–363

Hagberg, A., Schult, D., & Swart, P. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds)*, Pasadena, CA USA, p. 11–15

Humphries, M. & Gurney, K. (2008). Network 'small-world-ness': A quantitative method for determining canonical network equivalence (network 'small-world-ness'). *PLoS ONE*, 3(4), p.e0002051

Jeronimus, M., Westerveld, S., van Leeuwen, C., Bhulai, S. & van den Berg, D. (2017). Japanese Kanji characters are small-world connected through shared components. Red Hook, NY USA. *IARIA*

Lukatela, G. & Turvey, M. (1990). Phonemic similarity effects and prelexical phonology. *Memory & Cognition*, 18(2), 128-152

Marslen-Wilson, W. (1987). Functional parallelism in spoken word-recognition. *Cognition*, vol. 25(1), p. 71-102.

Motter, A., de Moura, A., Lai, Y., & Dasgupta, P. (2002). Topology of the conceptual network of language. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 65(6, Pt. 2), p(065102)

Pinker, S. (1998).. Words and rules. *Lingua*, 106(1), p. 219-242

R Core Team (2017). R: A language and environment for statistical computing. *R Foundation for Statistical Computing*, Vienna, Austria

Vitevitch, M. (2008). What can graph theory tell us about word learning and lexical retrieval? *Journal of Speech, Language, and Hearing Research*, vol. 51, p. 408–422

Wasserman, S. & Faust, K. (1994). *Social network analysis: methods and applications*. Cambridge, UK. Cambridge University Press

Watts, D. & Strogatz, S. (1998). Collective dynamics of small-world networks*. Nature*, vol. 393, p. 440–442

Appendix 1: Script for linking phonological neighbours

```
# lexicon.py
# Casper van Velzen, University of Amsterdam, 11030275
# creates a network of words linked if they are phonological neighbours
# requires the networkX package to work

import networkx as nx
import csv

# file with phonological transcriptions
inputfile = 'transcriptions.csv'

# initiate empty graph
network = nx.Graph()
prev_word = ""

# load file with phonological transcritions
with open(inputfile, encoding='utf-8') as csvfile:
	# create node in network for every word
	for i, word in enumerate(csvfile):
		# test for duplicate words
		if word != prev_word:
			l = list(word.rstrip())
			for n, char in enumerate(l):
				# count long vowels as one phoneme
				if char == ":":
					l[n-1] = l[n-1] + l[n]
					del l[n]
			network.add_node(i, ipa=list(word.rstrip()))
		prev_word = word


# check for minimal pairs
for j in range(len(network)):
	for k in range(j + 1, len(network)):
		word1 = network.nodes[j]['ipa']
		word2 = network.nodes[k]['ipa']
		lenght1 = len(word1)
```

```
lenght2 = len(word2)

# check for minimal pair when substituting phoneme
if lenght1 == lenght2:
        ndiff = 0
        for n, phoneme in enumerate(word1):
                if word2[n]!=phoneme:
                        ndiff+=1
        if ndiff == 1:
                network.add_edge(j, k)
                print(str(word1) + ' ' + str(word2))

# check for minimal pair when subtracting phoneme
elif lenght1 - 1 ==  lenght2:
        ndiff = 0
        # remove every phoneme one by one to check for match
        for n, phoneme in enumerate(word1):
                # convert str to list so it is mutable
                l = list(word1)
                # remove one phoneme
                del(l[n])
                newstr = "".join(l)

                if newstr == word2:
                        ndiff+=1

        if ndiff == 1:
                network.add_edge(j, k)
                print(word1 + ' ' + word2)

# check for minimal pair when adding phoneme
elif lenght2 - 1 ==  lenght1:
        ndiff = 0
        # remove every phoneme one by one to check for match
        for n, phoneme in enumerate(word2):
                # convert str to list so it is mutable
                l = list(word2)
                # remove one phoneme
                del(l[n])
                newstr = "".join(l)

                if newstr == word1:
                        ndiff+=1

        if ndiff == 1:
                network.add_edge(j, k)
                print(word1 + ' ' + word2)
```

```
edges = network.number_of_edges()
print("Number of nodes: " + str(nx.number_of_nodes(network)))
print(edges)

# get the biggest connected component
biggest_comp = max(nx.connected_component_subgraphs(network), key=len)
nodes = nx.number_of_nodes(biggest_comp)
edges = biggest_comp.number_of_edges()
apl = nx.average_shortest_path_length(biggest_comp)
cc = nx.average_clustering(biggest_comp)
print("Biggest comp nodes: " + str(nodes))
print("Number of edges: " + str(edges))
print("APL: " + str(apl))
print("CC: " + str(cc))
```

Appendix 2: Script for ER-randomisation

```
# erdos_renyi.py
# Casper van Velzen, University of Amsterdam, 11030275
# generates random graphs and writes relevant statistics to a csv
# requires the networkX package to work

import networkx as nx
import csv

# number of nodes
n = 652
# chance to connect two nodes (nr of connections / max number of connectons)
p = 5841 / ((n * (n - 1)) / 2)
print(p)
nr_of_graphs = 1000
outputfile = 'er_phonoclx.csv'

data = {}

with open(outputfile, 'w', newline='') as csv_file:
        writer = csv.writer(csv_file)

        for i in range(nr_of_graphs):
                print("Graph nr: " + str(i))
                G = nx.fast_gnp_random_graph(n, p, seed=i)
                print("Number of nodes (full): " + str(nx.number_of_nodes(G)))
                print("Number of edges (full): " + str(nx.number_of_edges(G)))
                data[i] = {}

                biggest_comp = max(nx.connected_component_subgraphs(G), key=len)
                nodes = nx.number_of_nodes(biggest_comp)
```

```
print("Biggest comp nodes: " + str(nodes))
data[i]["Big_comp_nodes"] = nodes

edges = nx.number_of_edges(biggest_comp)
print("Biggest comp edges: " + str(edges))
data[i]["Big_comp_edges"] = edges

apl = nx.average_shortest_path_length(biggest_comp)
print("APL: " + str(apl))
data[i]["APL"] = apl

cc = nx.average_clustering(biggest_comp)
print("CC: " + str(cc))
data[i]["CC"] = cc

print("***")

row = []
for key, value in data[i].items():
        row.append(value)

writer.writerow(row)
```