|  |  |  |
| --- | --- | --- |
| The post [Analyse Site Structure and Internal Linking Structure with igraph](https://lucidmanager.org/analyse-site-structure/) appeared first on [The Lucid Manager](https://lucidmanager.org/).  One of the essential parts of Search Engine Optimisation (SEO) is a logical site structure. Site structure relates to how the pages of a website link to each other. A well-designed structure of internal links improves the user’s experience. It also helps web crawlers to find their way through the website. This article shows how to use the igraph software to analyse site structure.  The iGraph package is a versatile library that can analyse many different types of data, some examples:   * [The Trump business network](https://lucidmanager.org/trumpworld-analysis/) * [Literature review](https://lucidmanager.org/discourse-network-analysis-literature-review/) * [Enron email traffic](https://lucidmanager.org/analyse-enron-corpus/)   The Lucid Manager website has been around for ten years now, and the content has grown organically over time, which is a reflection of my changing interests and evolving career. Although I use categories and tags to organise the articles, internal linking between posts is a bit haphazard. This post shows how I extracted the relevant data from the WordPress database and visualised the internal links with the iGraph package. This analysis shows that my website needs a bit of rework to improve its jungle of internal links to provide you and search engines a better experience. You can view the R code to investigate site structure below or download the latest version from my [GitHub](https://github.com/pprevos/r.prevos.net/tree/master/Miscellaneous/Wordpress) repository.   |  | | --- | | Literature Review. | | Donald Trump’s business network. |

**Site Structure**

A website needs to have a [structure](https://yoast.com/site-structure-the-ultimate-guide/) to prevent it from being a loose collecting of articles. Internal links on a website organise the available information through taxonomy, and these links provide context to the text by referring to related pages. The taxonomy consists of menus, breadcrumbs, categories, tags and other structural elements. Contextual links are located in the body of text of each article and provide additional or related information to the reader.

WordPress automatically adds links to categories and tags and the creates an ordered structure. Body text links are produced much more organically and can therefore quickly turn into a chaotic jungle of relationships. WordPress has several plugins that help with site structure, but not of these tools provide a complete overview of the structure in the internal links.

The taxonomy provides an automated structure of the website by categorising articles. The internal linking structure is organic and needs some further consideration. This analysis focuses on the organic links within the body of the text to visualise and analyse the internal linking structure.

**Extracting WordPress Data**

The Lucid Manager is created with WordPress. This system stores the body text of all articles in the

wp\_posts

table in the database. To analyse site structure, we need the following fields:  
–

post\_name

(slug)  
–

post\_content

(full text)  
–

post\_type

(only posts)  
–

post\_status

(only published posts)

The post type is in most instances either a page or a post, but can also be attachments and so on. The post name (slug) links posts together, and the post content contains the full text of the article in HTML.

Several methods are available to extract the required data from the WordPress database. The easiest way is to use a plugin such as [WP All Export](http://www.wpallimport.com/export/). You can drag-and-drop the required fields or run a SQL query. The second method is to log in to the phpMyAdmin on your cPanel and run the relevant queries.

The third method involves using the [RMySQL](https://cran.r-project.org/web/packages/RMySQL/index.html) package to download data from the site directly. You will need to create a new user with limited access rights and open the database to your IP address.

The first step reads the data from the database and selects all published posts. The credentials are read from the

dbconnect.R

file, so I don’t publish sensitive data on this website. The table names will be different if you use a WordPress network. Thanks to ‘phaskat’ on the [WordPress StackExchange](https://wordpress.stackexchange.com/questions/333163/i-would-like-some-help-wth-an-sql-query-to-link-posts-with-categories/333189#333189) for helping with the queries. Please note that when using a WordPress network (as I do), you will need to replace the wp prefix in the table name with the relevant string.

## Load data (CSV exported from wp\_posts table in MySQL database)

library(tidyverse)

library(tidytext)

library(igraph)

library(RMySQL)

library(RColorBrewer)

## Download data using RmySQL

source("dbconnect.R")

## lucidmanager <- dbConnect(MySQL(),

## user = "username",

## password = "Password1",

## dbname = "databe name",

## host = "URL")

posts.query <- "SELECT p.post\_name, p.post\_content, t.name

FROM wp\_posts p

JOIN wp\_term\_relationships tr ON (tr.object\_id = p.ID)

JOIN wp\_term\_taxonomy tt ON (tt.term\_taxonomy\_id = tr.term\_taxonomy\_id )

JOIN wp\_terms t ON (t.term\_id = tt.term\_id)

WHERE

p.post\_type='post'

AND

p.post\_status='publish'

AND

tt.taxonomy = 'category'"

pages.query <- "SELECT post\_name, post\_content

FROM

wp\_posts

WHERE

post\_type='page'

AND

post\_status='publish'"

posts.data <- dbSendQuery(lucidmanager, posts.query)

posts <- fetch(posts.data, n=-1)

pages.data <- dbSendQuery(lucidmanager, pages.query)

pages <- fetch(pages.data, n=-1)

dbDisconnect(lucidmanager)

pages$name = "Page"

content <- rbind(posts, pages)

as\_tibble(content)

**Converting the data to a network**

The second step uses the [tidytext package](https://www.tidytextmining.com/) to convert the texts of the articles into tokens. In this case, a token is defined as any set of characters between spaces. In the default setting, the tokenisation function also splits words at dashes, which is not helpful because I want to preserve the hyperlinks as one token.

The third step uses regular expressions to find internal links and only retain the slug. The slug is the text after the website name. The

post\_name

field in the database contains the slug for the post itself.

These steps result in a table that shows how all slugs link to each other. In network analysis, this is an [adjacency list](https://en.wikipedia.org/wiki/Adjacency_list) as each line represents a relationship in the network.

links <- content %>%

unnest\_tokens(word, post\_content,

token = stringr::str\_split, pattern = " ") %>%

filter(grepl(".+lucidmanager.org/", word)) %>%

mutate(link = gsub(".+lucidmanager.org/", "", word),

link = gsub("/.+", "", link)) %>%

filter(link != "tag" & link != "wp-content" & link != "author") %>%

select(category = name, post\_name, link)

tibble(links)

**Analysing Site Structure with iGraph**

Network analysis is the perfect tool to analyse site structure because each post on the website is a node and a link between two articles is a graph edge (arrow).

The Lucid Manager currently discusses strategic and fun data analysis, but in the past, I wrote articles about water utility marketing and critical perspectives on management theories. The questions I like to answer with this analysis are:  
– Are all pages connected to each other in one or more steps?  
– What is the most linked page?  
– Which pages link to themselves?  
– Are there duplicated links within one post?

The iGraph software transforms the link table to a network. Any posts and pages without any links (solitary pages) are added separately. Each post is also given a colour related to its category, and pages are a separate colour.

This analysis shows that the internal link structure contains several sub networks, which are groups of pages that are only linked to each other. This graph will help in to identify more linking opportunities to dissolve sub networks.

The [degree function](https://igraph.org/r/doc/degree.html) in iGraph determines the number of adjacent edges of each graph. The degree function can specify incoming and outgoing links. My article about service quality in water utilities is the most linked post on this site. In network analysis, this is the node with the highest degree of linking.

The loops in the diagram indicate self-referencing articles. The [which\_loop](https://igraph.org/r/doc/which_multiple.html) function identifies which arrows have the same start and end. The E function shows the list of edges, which shows that four pages on my website refer to themselves.

Lastly, the [which\_multiple](https://igraph.org/r/doc/which_multiple.html) function identifies any duplicated edges in the graph. Within this context, these are pages that contain the same link more than once. The Lucid Manager website has seventeen instances of duplicated links.

This first phase gives me plenty of homework to improve the internal linking structure of this website.

Visualising website structure.

solitary <- c(posts$post\_name, pages$post\_name)[!(c(posts$post\_name, pages$post\_name) %in%

unique(c(links$post\_name, links$link)))]

network <- select(links, post\_name, link) %>%

graph\_from\_data\_frame(directed = TRUE) +

vertices(solitary)

colour <- tibble(post\_name = V(network)$name) %>%

left\_join(content) %>%

mutate(name = factor(name),

colour = brewer.pal(length(unique(content$name)), "Set2")[name]) %>%

select(-post\_content)

V(network)$color <- as.vector(colour$colour)

par(mar = rep(0, 4))

plot(network,

layout = layout.fruchterman.reingold,

vertex.label.cex = .7,

vertex.size = degree(network),

vertex.label.color = "black",

vertex.frame.color = NA,

edge.arrow.size = .2,

edge.color = "darkgray"

)

## Diagnose network

which.max(degree(network, mode = "in"))

E(network)[which\_loop(network)]

E(network)[which\_multiple(network)]

**Future improvements**

This analysis is a great step to start systematically analysing organic website structure. Several other techniques help to understand the structure of a website further. Community detection and the inclusion of tags and categories in the data can provide additional context.

SEO experts might find this concept useful. Perhaps somebody else can develop a WordPress plugin to provide this visualisation.