Web-Based PageRank Visualizer and the Influence of Content Quality Metrics on this Algorithm

Author: Diogo Silva

Date: 12/06/2020

Index

1 Context and Objectives

This work has been constructed for the course of *Aspetos Profissionais de Engenharia de Software* at Universidade de Aveiro proceeding a proposal by the head teacher Manuel de Oliveira Duarte. The base idea consisted in the development of a tool that could be used in a pedagogical fashion to instruct users about the functioning of the **PageRank** algorithm and its many nuances. The effort boiled down to the construction of an appealing graphical environment that would allow anyone, no matter the level of knowledge on the subject, to explore and observe how the PageRank algorithm works and shifts in accordance to changes made to a network. Furthermore, it was also asked that an additional algorithm was also implemented - a modification to the baseline PageRank algorithm that would take into consideration the influence of the quality of a page's quality on its score, inspired by similar principles of market adoption evolution overtime when considering the available products' quality and initial market share. In this vein, the main goals of this work are:

- Create a tool that allowed for the exploration and teaching of several factors and gradations of the PageRank algorithm, alongside concepts such as Random Surfers, HyperLink Matrices and anomalies.
- Create a modification of the PageRank algorithm inspired by algorithms and principles of market evolution overtime influenced by the quality of the product.
- Be as pedagogical as possible in order to allow the tool to be used as a learning aid.

It should be stated that the initial proposal was to have the software be developed in MatLab, utilizing their recent Simulink framework, but for sake of having a broader reach and ease of access, the tool was created and designed as a Web Application, having been built in React JS. The finalized version is now available for public access on this link.

2 Brief Introduction to PageRank

2.1 The Algorithm

The aim of this report isn't to provide a full explanation of the **PageRank algorithms**. There are other pieces of work that already do this and do it better [1] [4] [3]. As such we will briefly mention how the PageRank works whilst diverting our focus to explaining how this mathematical algorithm was made transformed into programming code. It should also be mentioned that, whilst all code was written in JavaScript, we will presenting a generic view of the code without delving too much into the nuances of this language.

For the uninitiated, the PageRank algorithm is an iterative mathematical algorithm that aims to rank pages in a network based on how many other pages reference/link to it. It was developed by Google in the late 1990's and has been one of the weightlifters that allowed this company to raise into such prominence. The Google Search Engine applied this algorithm in a large-scale to the entire web, presenting pages sorted by their **page ranks**, giving more importance to showing the higher rated ones, whilst pushing the lowest rated ones down. PageRank assumes and idealizes that the web behaves and takes the shape of a **directed graph** (i.e a set of nodes, which represent pages, connected by edges that have a direction, which represent links between the pages).

As aforementioned, the PageRank algorithm is iterative. At iteration 0, each page is assigned an equally distributed Page Rank value (which range from 0 to 1). This is achieved by the following equation, where $pr(x)_0$ represents the page rank of any given page in the network at iteration 0 and n is the total number of

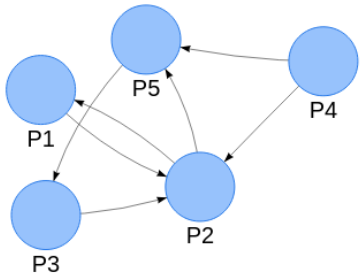


Figure 1: An example of a possible network graph. This screenshot was taken using the PageRank-Visualizer Web-App, but the example configuration has been inspired by the network present in António Teixeira's MPEI Slides [4]

nodes in our network. At some point the page ranks will converge to a consistent value, at which point we determine the PageRank algorithm to have **stabilized**

$$pr(x)_0 = \frac{1}{n} \qquad \text{(Equation 2.1)}$$

For the following iterations we can compute the page rank of each page in several ways, but the one we chose to focus one was the HyperLink method, which can be given by the following equation, where *pr* is the *nx1* matrix of page rank values (where each cell corresponds to the page rank of one page in the network) and *H* is the HyperLink Matrix, which we will further explain in the next subsection.

$$pr_{k+1} = pr_k * H \qquad \text{(Equation 2.2)}$$

Our programmed PageRank algorithm is hence given by the following set of simplified instructions (Algorithm

1.).

Algorithm 1: How to compute the PageRank values for all pages on a given iteration

Result: An updated array of each page's page ranks

begin

$start = []$;
$n \leftarrow numberofpages$;
$InitialVal \leftarrow \frac{1}{n}$;
for *node* ***in*** *nodes in network* **do**
$start.insert(node : node.id, pr : InitialVal)$;
end
if *new hyperlink matrix* **then**
$ComputeHyperLinkMatrix(allNodes)$
end
$curValArray = []$;
for *node* ***in*** *allNodes* **do**
$curValArray.insert(node.pr)$;
end
$curValArray = []$;
for $j = 0$; $j < start.length$; $j++$ **do**
$curValArray \leftarrow curValArray * HyperLinkMatrix$;
end

end

2.2 HyperLink Matrix

The Hyperlink Matrix, which has been referenced on the previous section, consists in an *nxn* dimensions matrix where each cell (*i,j*) (where i corresponds to the row and j the column), contains the probability of going from page *i* to page *j*. The probability is equally divided between all pages the outgoing page links to, so for example, if page A links to two others, B and C, then it is said that the probability of going from A to B is 50% (or 0.5) and of going from A to C is also 50% (or 0.5). Each cell's value can, therefore, be computed as exemplified below, where $H_{i,j}$ is the cell (*i,j*) on the HyperLink Matrix and *m* is the total number of outgoing links from page *i*.

$$H_{i,j} = \begin{cases} \frac{1}{m}, & \text{if there exists a link from node i to j} \\ 0, & \text{otherwise} \end{cases} \qquad \text{(Equation 2.3)}$$

It should be stated that, whilst in the real web some pages might link to themselves (for example by having the page divided in sections and containing links between these), neither the PageRank nor the HyperLink matrix accommodate for these situations as these technically are not considered outgoing links, but simple scrolls within the same page. It should also be said that the nodes represent pages, not websites and as such, whilst some pages in a website may link to others in the same base URL, they're in fact linking to different pages. This means that in our network graph we may have multiple pages that belong to the same website, but are considered individual entities.

The HyperLink Matrices work on most types of networks, but certain anomalies may cause them to behave

unexpectedly. For example, one of the main properties of this matrix is that all rows' values should sum up to 1 (or to 100% when working with probabilities rather than decimals). This, however, is not the case when **Dead End anomalies** occur. We will explain these further in the next section, but it should be referenced that an HyperLink matrix that treats both **Dead Ends** and **Spider Traps** is possible, and the following equation is capable of producing such a matrix. This equation achieves what we call the **Google Matrix**. *G* corresponds to the Google Matrix, θ is called the dampening factor, and it dictates the probability of a user traveling the web traveling via following a link versus by making a disconnected jump (which would be accomplished through the usage of inputting the URL directly or using bookmarks), *Ones(l,r)* corresponds to a matrix of dimensions (*l,r*) filled with 1's and *w* is a (nx1) dimensions matrix with 0s on all nodes, except for dead end nodes which have a 1.

$$G = \theta \cdot (H + \frac{1}{n} \cdot w \cdot Ones(1, n)) + (1 - \theta) \cdot (\frac{1}{n} \cdot Ones(n, n)) \qquad \text{(Equation 2.4)}$$

From a programming standpoint, the algorithm could be showcased as the following Algorithm 2.

Algorithm 2: How to compute the HyperLink Matrix

Result: An HyperLink Matrix

begin

$n \leftarrow numberofnodes$;
$hMatrix \leftarrow matrix(zeros([n, n]))$;
$InitialVal \leftarrow \frac{1}{n}$;
for *node* ***in*** *nodes in network* **do**
$relations \leftarrow []$;
for *relation* ***in*** *relations in network* **do**
if $relation.from \neq node.id$ **then**
$continue$;
end
for *node2* ***in*** *nodes in network* **do**
if *node2.id == relation.to* **then**
$relations.insert(node2)$
end
end
end
$value \leftarrow 0$;
if $relations.length \neq 0$ **then**
$value \leftarrow \frac{1}{relations.length}$;
end
for *relation* ***in*** $relations$ **do**
$hMatrix[node.index, relation.index] \leftarrow value)$;
end
end

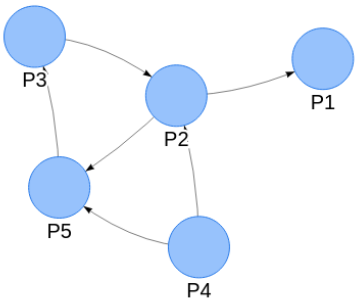
end

2.3 Anomalies

Lets now divert our attention to the two main anomalies that might happen depending on the network graph's configuration.

2.3.1 Dead Ends

Dead Ends are anomalies caused by the presence of pages that posses no outgoing links. This will cause any unfortunate user that happens to travel to one to be unable to go to any other page being thereafter forever stuck in that dead end. These occurrences have the particularity of being easily identifiable in the HyperLink Matrix as they will cause it to have the rows corresponding to these dead end (or **dangling**) pages filled with nothing but 0s.



HyperLink Matrix

Page/Page	P1	P2	P3	P4	P5
P1	0	0	0	0	0
P2	0.5	0	0	0	0.5
P3	0	1	0	0	0
P4	0	0.5	0	0	0.5
P5	0	0	1	0	0

Figure 2: In this network the pages P1 is a dead end since it has no outgoing links. As can be seen on the picture to the right, the HyperLink matrix row corresponding to P1 is filled with nothing but 0s. This screenshot was taken using the PageRank-Visualizer Web-App, but the example configuration has been inspired by the network present in António Teixeira's MPEI Slides [4]

A way to solve the problem of users getting stuck on dangling nodes can be simply resolved by giving every one of these nodes an equal probability of being proceeded by a jump to any node in the network. This is expressed by the following equation in which $\hat{H}$ corresponds to the new HyperLink matrix and *w* is a (nx1) dimensions matrix with 0s on all nodes, except for dead end nodes which have a 1 [3].

$$\hat{H} = H \cdot \frac{1}{n} \cdot w \cdot Ones(1, n) \qquad \text{(Equation 2.5)}$$

Algorithms-wise this isn't too hard to achieve as can be seen on Algorithm 3.

Algorithm 3: How to recompute the HyperLink Matrix to avoid Dead Ends

Result: An updated HyperLink Matrix capable of treating Dead Ends

begin

$n \leftarrow$ number of nodes;
$hMatrix \leftarrow$ current HyperLink Matrix;

$initialVal \leftarrow \frac{1}{n}$;
$onesArray \leftarrow ones(1, n)$;
$danglers \leftarrow zeros(n, 1)$;
for $i \leftarrow 0 \ ; 1 < n \ ; i++$ **do**
  $rowZeros \leftarrow true$;
  for $j \leftarrow 0; j < n; j++$ **do**
    if $hMatrix(i, j) \neq 0$ **then**
      $rowZeros \leftarrow false$;
      $break$;
    end
  end
  if $rowZeros == true$ **then**
    $danglers[i, 0] \leftarrow 1$;
  end
end
$multi \leftarrow danglers \cdot onesArray$;
$finalMatrix \leftarrow hMatrix + (initialVal \cdot multi)$;

end

2.3.2 Spider Traps

Spider Traps consist in a set of nodes that create a loop between each other due to not having outgoing links that would allow a user to escape traveling between the same pages. Unlike Dead Ends which are easily identified just by looking at the HyperLink Matrix, Spider Traps are a bit more nuanced. However, they're just as easily treatable.

To solve these anomalies all we have to do is apply a **dampening value**, θ, which dictates whats the probability of a user, in their next jump, traveling to a page by following an outgoing link in their current page versus simply making a disconnected jump, by inputting a URL or using a bookmark. As such we have the following equation.

$$\hat{H} = \theta \cdot H + (1 - \theta) \cdot (\frac{1}{n} \cdot Ones(n, n)) \quad \text{(Equation 2.6)}$$

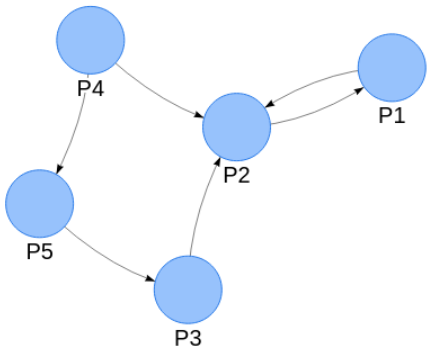


Figure 3: In this network the pages P1 and P2 constitute a Spider Trap, after a user reaches P2 they can only then travel to P1, and after reaching P1 they can only jump back to P2, hence being stuck in an infinite loop between those two pages. This screenshot was taken using the PageRank-Visualizer Web-App, but the example configuration has been inspired by the network present in António Teixeira's MPEI Slides [4]

In terms of pseudo-code this is also very much achievable using the following algorithm 4.:

Algorithm 4: How to recompute the HyperLink Matrix to avoid Spider Traps

Result: An updated HyperLink Matrix capable of treating Spider Traps

begin

$n \leftarrow$ number of nodes;
$hMatrix \leftarrow$ current HyperLink Matrix;

$initialVal \leftarrow \frac{1}{n}$;
$randomJump \leftarrow initialVal \cdot Ones(n, n)$;
$continuing \leftarrow \theta \cdot hMatrix$;
$jumpProb \leftarrow 1 - \theta$;

$randomJump \leftarrow jumpProb \cdot randomJump$;
$finalMatrix \leftarrow continuing + randomJump$;

end

2.4 Random Surfers

The final concept we will be explaining about the PageRank algorithm is the one of **Random Surfers**. This is the generic name we give to a simulation that aims at imitating the behavior of a normal human user when traveling the web. According to PageRank's theory [1] [3] [4] , the following are the main common behaviours of users when in the internet:

- They start on a random website within the network
- They travel to a new page either by clicking on an outgoing link present within their current page or by using shortcuts (such as inputting a URL or using a bookmark)

This way of interpreting users is what lead us to many of the concepts discussed in the prior sections and is

undoubtedly the basis for the PageRank algorithm. As such, a way to display this behaviour could be achieved using the following Algorithm 5., based around a **random number generator**.

Algorithm 5: How to pick the next node for the Random Surfer to jump to

Result: The next node the Random Surfer will jump to

```
begin
    pageRankValues ← Array of current PageRank Values sorted from lowest to highest rank;
    curPage ← null if this is the surfer's first jump, current node otherwise;
    samePageJumps ← true if we want to allow same page jumps, false otherwise;
    disconnectedJumps ← true if we want to allow same page jumps, false otherwise;
    for attempt ← 0; attempt < 500; attempt++ do
        randomNumber ← random number between 0 and 1;
        threshold ← 0;
        winner ← null;
        for i ← 0; i < pageRankValues.length; i++ do
            threshold+ = pageRankValues[i].pageRank;
            if threshold > randomNumber then
                winner = pageRankValues[i];
                break;
            end
        end
        if winner! = null then
            violation ← false;
            if curPage ≠ null and winner.id == curPage.id and not samePageJumps then
                violation ← true
            end
            if curPage ≠ null and not disconnectedJumps then
                containsLink ← false;
                id ← curPage.id;
                for relation in all relations do
                    if relation.from == id and relation.to == winner.id then
                        containsLink ← true;
                        break;
                    end
                end
                if not contins Link then
                    violation ← true;
                end
            end
            if not violation then
                return winner
            end
        end
        return null
    end
end
```

This algorithm will attempt to find a viable jump (attempting to at maximum 500 times) and even allows the user to choose whether they want to allow or disallow jumps to the same page as the current or between disconnected pages.

3 The Quality PageRank

3.1 Introduction to the influence of quality

The PageRank presented in the prior section is the *de facto* base algorithm made and popularized by Google and their search engine, however, it's not hard to realize that the way it scores pages might present some disparity to what pages users migth be the most interested in on the internet. Truly, the more other pages link to a page, the more reputable it probably is, as well as the more entry points there are into it, hence it's logical that it'll generate more traffic. But one does ponder, how would that traffic, and alongside it, the page ranking, shift in accordance to the **quality** of the actual content within the page? Similar studies have been made on the topic of market adoption in accordance to a products quality [5]. With some small adaptations we could very easily morph the PageRank algorithm with these other ones, producing what we dubbed as the **Quality PageRank Algorithm** which takes into consideration the normal page rank at the current iteration, and applies equations of market adoption to produce a new score that attempts to simulate how popular a page would be, not just due to how many others link to it, but also due to how good the quality of its content is.

3.2 Explanation of the Algorithm

To achieve this, we first assign each page a quantitative **quality value**. This value should be a positive integer of any magnitude, but for purposes of simplicity, our algorithms assign an initial quality value of **10** to any page that is added to the network. Similarly to the regular PageRank algorithm, the quality influence on the market algorithms we adapted [5] are also iterative with the market share of each product, or in our case, the page rank value of each page, changing on each iteration in order to simulate an overtime gradual evolution. At iteration 0, every page's **quality page rank** is the same as their original page rank, as this points to the initial state of the market before users start shifting to a certain page due to its quality.

Overtime, however, and on each iteration, certain pages will see their ranks increase, whilst others will decrease, depending on their quality. This simulates how users are more attracted to pages with higher quality content, but don't all immediately flock to such pages, as these first need to start gaining traction and notoriety. The theory is that, at each iteration more and more users notice how good or bad each page's content is, and even pages which had a high page rank initially will start losing traffic if pages with better content show up, as they will dilute the market (or in our case, the network) attracting users to them.

Mathematically, first, and at each iteration, we have to compute the average quality of each page. This is done to represent the fact that when a user is deciding which page to visit, they will want to compare the quality of all pages, **relative** to the average quality of all pages on the network [5]. The average quality of the network's pages can be given by the following formula, where is the average quality, N is the number of pages in the network, Q_n is the quality of page n and QPR_n is the quality page rank of page n at the current iteration (which for iteration 0 corresponds to the original page rank values). Note that the original equations for the market evolution according to quality [5] requires that we divide the presented equation by the sum of the current market values, however since the sum of the page rank values at any given time should always be 1, this

would be a redundant step.

$$\bar{Q} = \sum_{n=0}^{N} Q_n \cdot QPR_n \quad \text{(Equation 3.1)}$$

On the last paragraph the word **relative** was emphasized. That was because, the next step after computing the current average quality is to, for each page, get its relative quality comparatively to the average, which, for each page, can be achieved with the following equation, where *QR* is the relative quality of any given page, and *Q* that page's quality value.

$$QR = \frac{Q}{\bar{Q}} \quad \text{(Equation 3.2)}$$

After this we have to compute what the market changes for the current iteration are. For this we require an additional constant called the **elasticity value**, E_Q. This **percentile** (0 to 100% or 0 to 1 in decimal) value represents the market's overall resistance to adopting to changes, i.e, how *elastic* users are to shift to and increase the traffic of pages with better quality and abandoning ones with less despite the original page rank values. The higher the elasticity value, the faster/in less iterations the **quality page rank** of pages with more quality will increase. The following equation calculates, for each page, the quality page rank change that will occur on the following iteration, this can be either a positive or negative value representing whether the page is seeing their quality page rank increased or decreased.

$$dQPR = E_Q \cdot (QR - 1) \quad \text{(Equation 3.3)}$$

For the current iteration, the new quality page rank value for each page is then given by the following final equation.

$$QPR(i) = QPR(i-1) + dQPR(t-1) \quad \text{(Equation 3.4)}$$

Applying this to code, we end up with the following Algorithm 6.

Algorithm 6: How to compute the Quality Page Rank values up to a given iteration

Result: An array of Quality PageRank Values

begin

$qPRValues \leftarrow$ current values of the page rank;

$noIterations \leftarrow$ number of iterations we want to compute;

for $i \leftarrow 0 \ ; i < noIterations \ ; i++$ **do**

$averageQuality \leftarrow 0$;

$marketSum \leftarrow 0$;

for $j \leftarrow 0 \ ; j < qPRValues.length \ ; j++$ **do**

$averageQuality \leftarrow qPRValues[j].current \cdot qPRValues[j].quality$;

$marketSum+ = qPRValues[j].current$;

end

$averageQuality \leftarrow \frac{averageQuality}{marketSum}$;

for $n \leftarrow 0 \ ; n < qPRValues.length \ ; n++$ **do**

$relativeQuality \leftarrow \frac{qPRValues[n].quality}{averageQuality}$;

$marketChange \leftarrow elasticity \cdot (relativeQuality - 1)$;

$qPRValues[n].current \leftarrow qPRValues[n].current + qPRValues[n].current \cdot marketChange$;

end

end

end

4 PageRank Visualizer

4.1 A Web-Based tool

Now that we have finished discussing some of the main baseline algorithms that we incorporated it's now time to turn our heads to our product at hand. The PageRank-Visualizer, as it was named, is a web-based application built to allow for the exploration and study of the PageRank algorithm, its phenomena and nuances, alongside presenting a modification made to it, the Quality PageRank, that aims at showcasing how the quality of the content in each page may affect their overall page rank.

This tool was built with the goal of being as pedagogic as possible since its main aim is to teach users about the functioning of these concepts featuring tool tips and a wide range of controls, whilst trying to be as clean and easy to use as possible with a minimalist UI and design. We also designed it to work universally on both phone devices and computers as well as on all mainstream browsers. The app has hence been made publicly available on this URL.

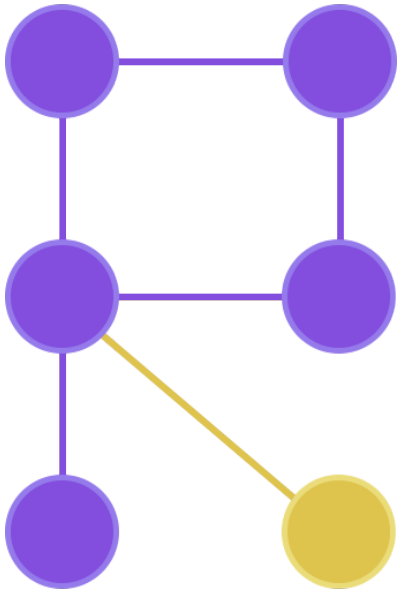


Figure 4: The PageRank-Visualizer Logo. This original logo shows a network configuration with the links between the nodes forming a P, when focusing on the magenta nodes, and an R, when taking into account the yellow one, the initials of PageRank.

4.2 Development and Deployment

The PageRank-Visualizer is a purely web-based app having been built utilizing the **React JS** [2] framework and written, appropriately, using the JavaScript programming language. As React is a widely modular tool set, we resorted to **MaterialUI** [11] due to it being one of the most widely used graphical component sets available. For the help messages that we display (which can be toggled on or off) were built using the **React-Toastify** [9] component made by Fadi Khadra, available on this public repository. Meanwhile the **React-Select** component [12], made by Jed Watson was also utilized for some of our form inputs.

Since JavaScript, by default, is a very poor (albeit versatile) language, we used **math.js**'s [10] library of mathematical functions and capabilities, specifically for their matrix calculus methods. This library was built and graciously made available for usage by Jos de Jong. As for displaying and controlling our physics-enabled network graph we used the graphical library Vis.js, or, more concretely, the **Vis.js Network** module [8].

As for the deployment, we published the app using **Heroku** as our host both due to prior experience and knowledge, but also because they offer completely free hosting with both American and European servers. Besides this, Heroku offers a really easy and nice deployment integration with GitHub [6] which allowed for a really easy publishing process.

All source code built has been made publicly available for consultation or adoption on this repository under the MIT License.

4.3 Features and Controls

Our PageRank-Visualizer contains a wide-range of features and capabilities that are presented as forthright as possible to the users. In this section we will be describing them succinctly in the hopes of enticing the reader to try out the tool for themselves. Our features can be divided into three groups: **Graph Controls**, **PageRank Controls** and **Quality PageRank Controls**. Before moving forward it should be explained that by default the page shows a rather simple network graph configuration, being the same configuration used in António Teixeira's slides [4]. This configuration was picked due to its simplicity but effectiveness in demonstrating the algorithms at work, as well as due to how easy it is to create anomalies in it (for example, removing the link from page P1 to P2 immediately creates a Dead End, whilst removing the link between P2 to P5 immediately creates a Spider Trap). A user can freely move both the graph and each node individually, with it being **physics enabled**. This effort was done because that way the user can shift the nodes and network into a position that they can better comprehend, as well as due to it being attention grabbing for new users seeing the page, which may increase their will to further explore the tool. Two additional controls also exist which don't fit into any of these groups. These two are the check boxes labeled *Show explanation tooltips* and *Use fractions*, the first of which will make it so tooltips get displayed when a process is running explaining what is happening, whilst the second changes the display from decimal values to fractions. Additionally, most controls posses a blue information icon that upon clicked will open an explanation of the algorithm or concept it pertains to.

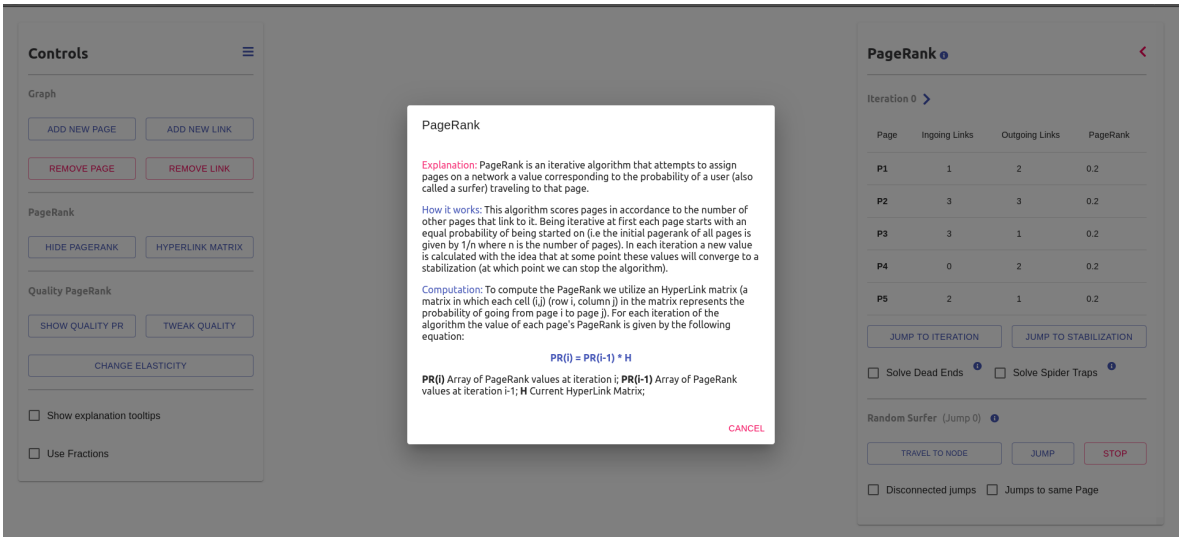


Figure 5: An example of a window that pops up when clicking an information icon, in this case the icon to the right of the PageRank title, which prompts a modal that offers a brief explanation of the PageRank algorithm.

4.3.1 Graph Controls

As the first of these, the **Graph Controls** group is the most prominent to what is being show on the screen, containing buttons that allow users to directly change and reconfigure the graph network. As such, these controls are placed on a left side menu (that can of course, be hidden by clicking a button), allowing it to be open at the same time as either the Quality PageRank or normal PageRank controls, so that the user may see the direct influence messing with the network may have on the results of these. These controls, all placed under the **Graph** section of the **Controls** side menu contain the ability to add and delete both nodes and links between them at will.

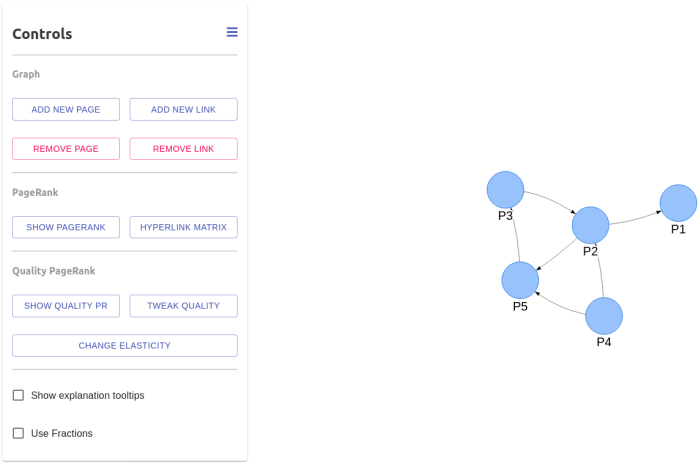


Figure 6: The Graph Controls, present under the Graph section of the Controls side menu, contain buttons that allow for the addition and deletion of both nodes and links. This entire side menu can be hidden by clicking on the blue *burger* icon.

Let's now go over the four buttons in this control-set

- **ADD NEW PAGE**
  Clicking on the button labeled ADD NEW PAGE brings up a *modal* (also known as *dialog*) window that allows a user to create a new name by inputting a name and pressing *ENTER*, or clicking the *CONFIRM* button. Note that if a user tries to create a new page with a name that is already taken by another, an error message will be shown and the new page will not be created (Figure 8.). This was done to prevent and protect users from creating pages with the same names, which would lead for a rather confusing network for them to analyse and see the processing of the algorithms happen. This way we enforce the good practice of giving each page an appropriate name. Note that if no name is specified an error message will also be shown stating that in order to create a page the user has to insert a name. If a name is given that has not yet been taken and nothing goes wrong, then the new node with the picked name will automatically be added to the network, albeit disconnected from any other nodes (as no links have been created between them yet) (Figure 9.).
- **REMOVE PAGE**
  Clicking the REMOVE PAGE button prompts a similar window where, instead of an input box we get a search-able select component that allows the user to pick from a list of all available nodes in the network for one they want to delete. A user can simply pick from this list and permanently delete the chosen node from the network as can be seen in Figure 10.
- **ADD NEW LINK**
  The button labeled ADD NEW LINK presents a modal with two form prompts. One labeled *TO* and one labeled *FROM*. Clicking each of these opens a search-able select containing all current nodes in the network. A user can then specify the page the link will be coming out of, by picking the *TO* node and which page the link points to, by picking the *FROM* page. If a user attempts to create a link from and to the same page the modal presents an error because, as aforementioned, links between the same page are something the PageRank algorithm doesn't take into account, therefore, we don't allow this (Figure

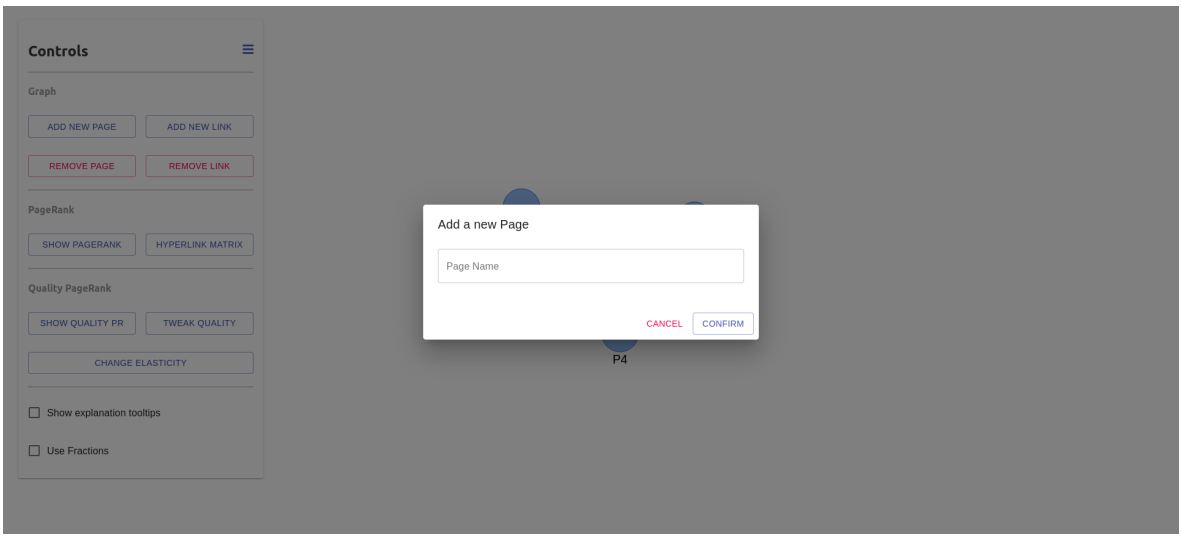


Figure 7: The modal that pops up when clicking the ADD NEW PAGE button. This modal prompts the user to input a valid name and click confirm in order to insert a new page into the network.

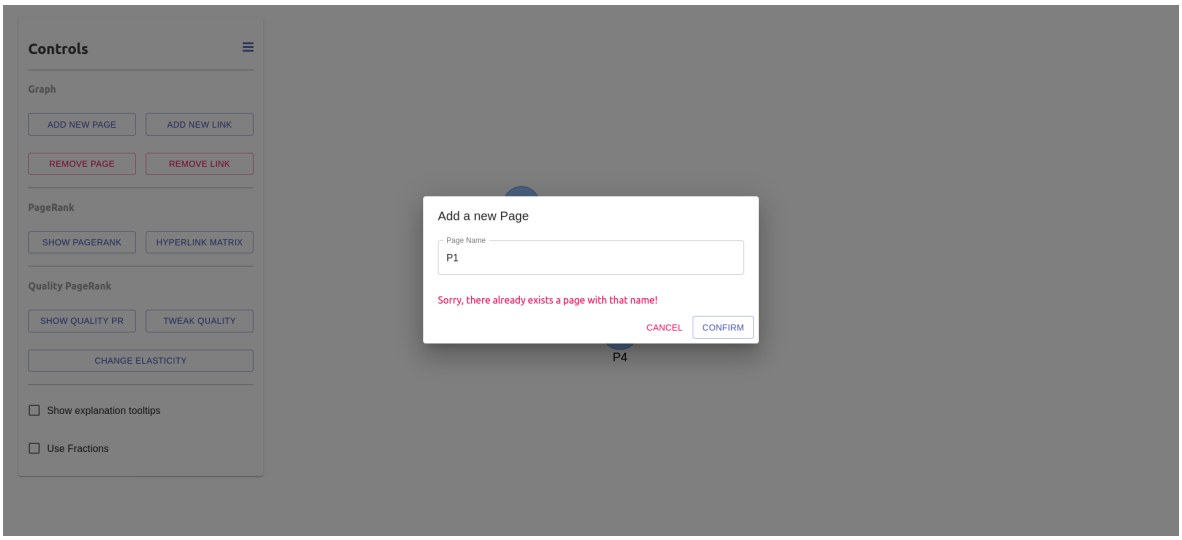


Figure 8: When trying to create a new page with the name that is already taken by another, then this error message will be presented to the user.

11.). Another error is also presented if one attempts to create a link that already exists (i.e there already is a link starting on the specified TO page and going to the specified FROM page), as the PageRank algorithm doesn't account for several duplicate links, and in fact what it cares for is whether one page links to another, not how many times it does so. After inputting a correct TO and FROM node, the link between them is automatically created and the network altered.

- **REMOVE LINK**
  The REMOVE LINK function works similarly to the one of REMOVE PAGE. A modal is also shown containing a select-able search bar that lists all of the links in the network, as can be seen in Figure 12..

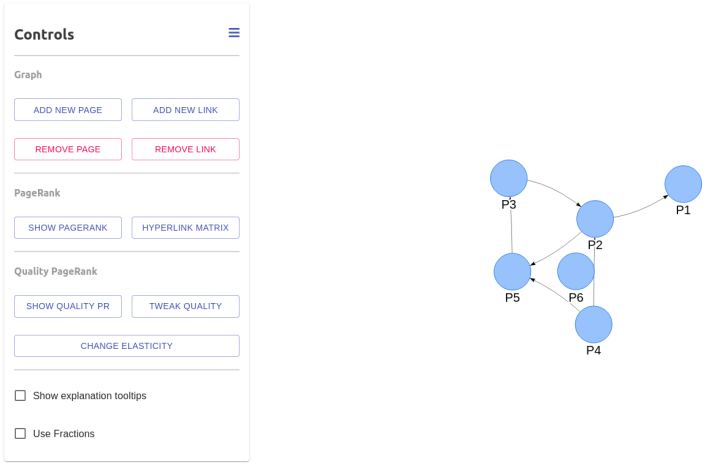


Figure 9: After successfully creating a new page, by naming it P6 and clicking CONFIRM, or pressing ENTER, it gets automatically placed amongst the other nodes

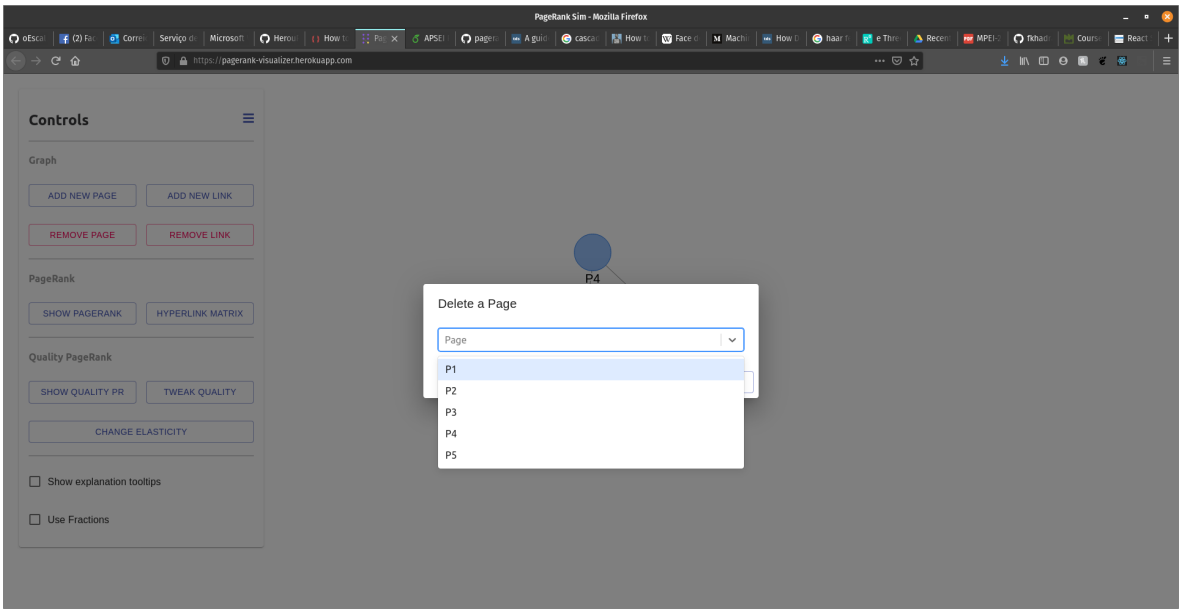


Figure 10: A user can select from all nodes in the network which they want to delete. After picking one clicking CONFIRM or pressing ENTER that node will be automatically removed from the network.

These links are presented with the labels $A \to B$, where A is the name of the outgoing node and B the in-going page.

4.3.2 PageRank Controls

Moving on we have the actual core of the web-app's concept. The PageRank Controls contain all interactions a user can have in terms of showing and controlling the PageRank algorithm with the current network

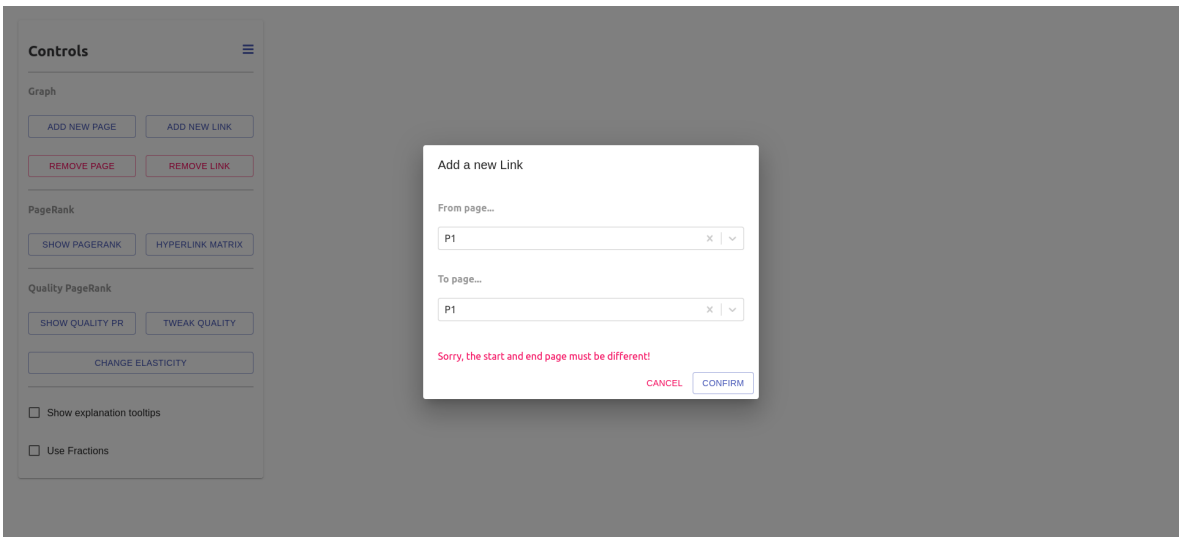


Figure 11: The error shown when trying to create a link from and to the same page.

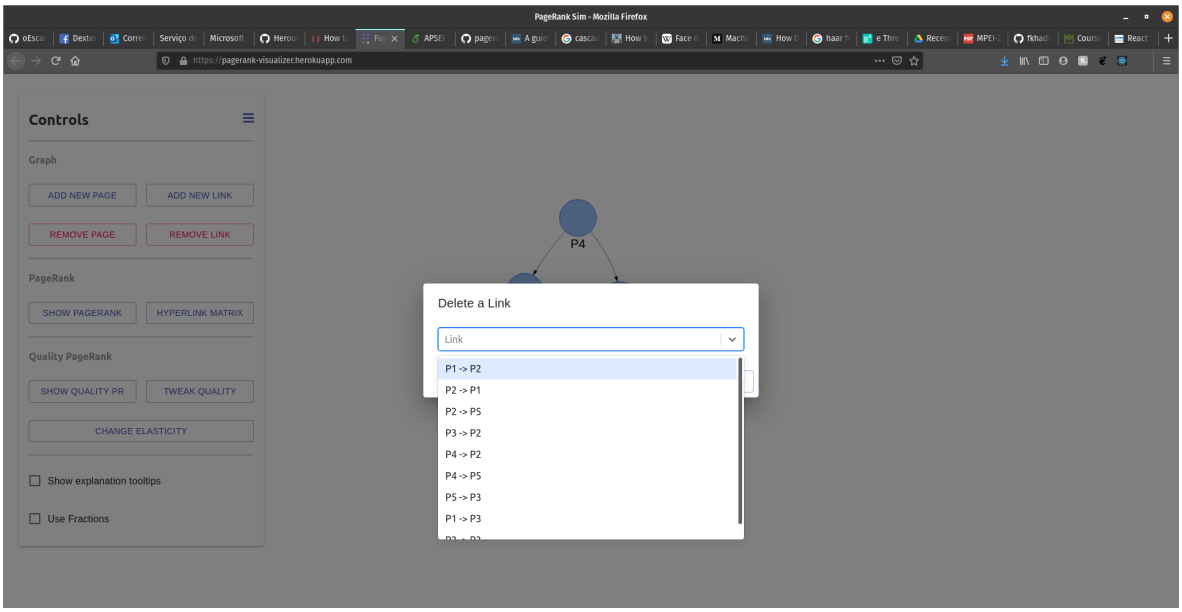


Figure 12: This modal allows a user to select from a list of all connections on the network and delete it.

configuration. This group can be further subdivided into three sub-groups: Menu Controls, PageRank Iteration Controls and Random Surfer controls. The first of these can be found on the left side menu under the PageRank section.

- **SHOW/HIDE PAGERANK**
  This simple button basically shows or hides the right side menu which is entirely dedicated to further PageRank controls. This organization and layout allowed for a clean interface with as less clutter as possible. It should be noted that the user can still change the network even whilst having this right menu open, however the Quality PageRank right side menu and PageRank right side menu cannot be opened

at the same time. This was done both for UI purposes as well as to avoid creating confusion between the two algorithms for users.

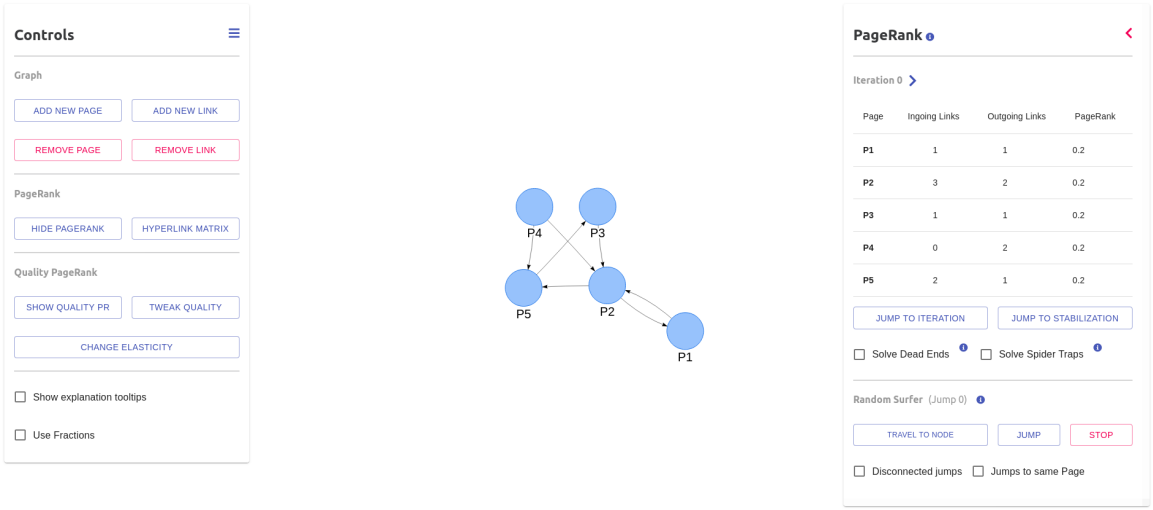


Figure 13: The right side PageRank menu that contains all other controls that pertain to the PageRank algorithm. When open the label on the button changes from SHOW PAGERANK to HIDE PAGERANK. This menu can then be closed either by re clicking the same button or by clicking the red *chevron* on the upper right corner.

- **HYPERLINK MATRIX**
  This button opens a modal window that displays the HyperLink matrix for the current network configurations, as simple as possible. This can be seen on Figure 14.

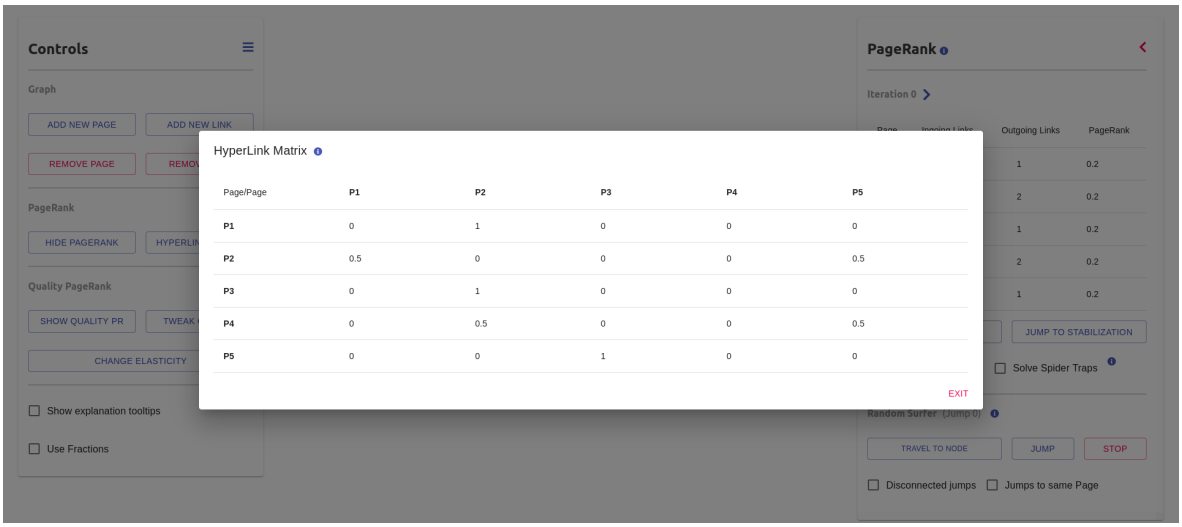


Figure 14: The modal window showing the current network's HyperLink Matrix

Now on the PageRank right side menu we firstly have the controls for advancing/regressing or jumping to a certain iteration. We also have two check boxes that change the HyperLink matrix in order to fix the

aforementioned anomalies (Dead Ends and Spider Traps). The main showcase in this section is the table that contains all Pages and their page ranks at the current iteration, as well as amount of in and out going links. This table is automatically updated when going to a new iteration of the PageRank.

- **NEXT/PREVIOUS ITERATION**
  A user can move to the next or previous iteration by clicking on the blue *chevrons* placed before and after the label *Iteration* , where is automatically updated to reflect what current iteration the PageRank algorithm is at. This can be seen on Figure 15.

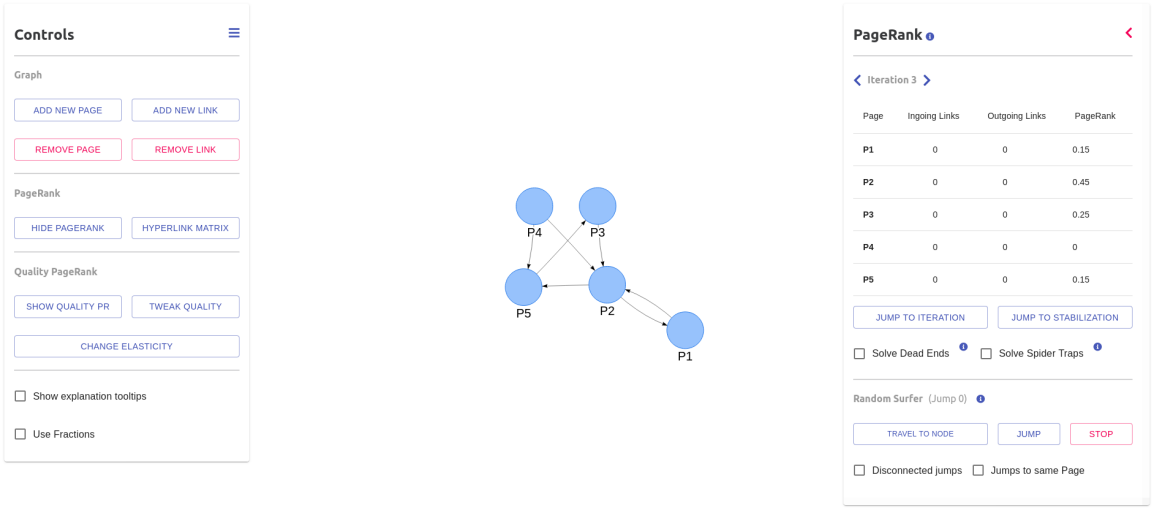


Figure 15: The table being updated as a user clicks on the right or left *chevron*

- **JUMP TO ITERATION**
  This button allows a user to jump to a specific iteration rather than going forward or backward one by one. Due to computational issues we only allow a user to jump to, at most, iteration 100000. Obviously we verify that the input iteration number is a valid non negative integer (as there are no such things as decimal or negative iterations) and present an error in case the users input an invalid number.
- **JUMP TO STABILIZATION**
  We've explained that eventually the page rank values converge into a stabilized value. As such, we've also included a button that allows the user to travel to the iteration where the values stabilized. However, and unfortunately, if stabilization can't be reached before iteration 100000, we simply travel to that iteration.
- **Solve Dead Ends Solve Spider Traps**
  Clicking these check boxes applies the modifications explained in section 2 to the HyperLink matrix that solves these two anomalies. If both options are ticked a message will also pop up, both on the menu and on the HyperLink modal window, stating that the HyperLink matrix has morphed into the **Google Matrix**. Clicking these buttons will, obviously, directly impact the current page ranks.

Finally, we also have controls that allow for the simulation of a random surfer. The surfer travels in accordance to the current PageRank's iterations value and the current node the surfer is on is showcased by highlighting the node in red.

- **TRAVEL TO NODE**

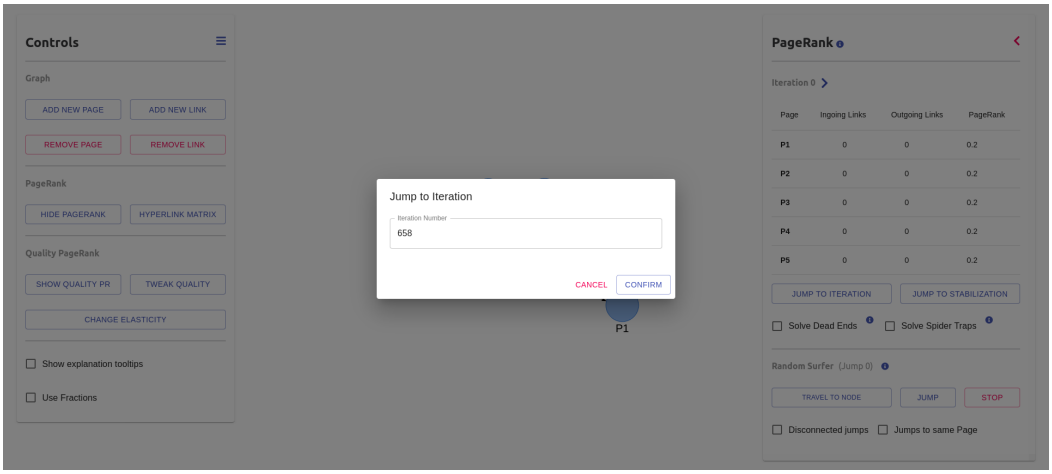


Figure 16: A user can input the PageRank iteration they want to travel to

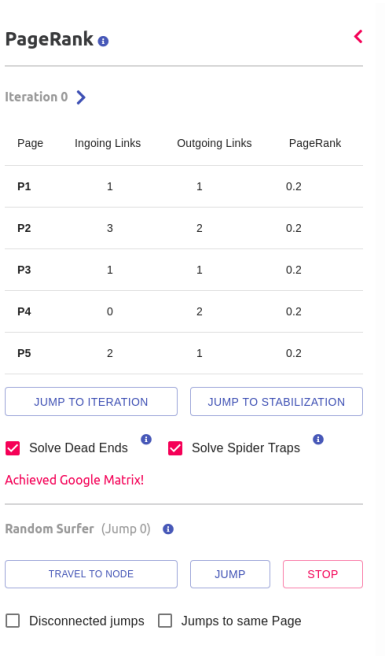


Figure 17: When solving both types of anomalies our HyperLink matrix becomes the Google Matrix.

Clicking this button allows the user to manually pick a node for the surfer to travel to

- **JUMP**
  This button makes the surfer perform a random jump. The choice of which node to jump to is given both by the current iteration's page rank values, as well as by whether the surfer is allowed to remain on the same page or jump to a node the current page doesn't connect to directly.
- **STOP**
  This button simply resets the jump count to 0 and removes the highlight on the surfer's current page, effectively removing the surfer from the network.

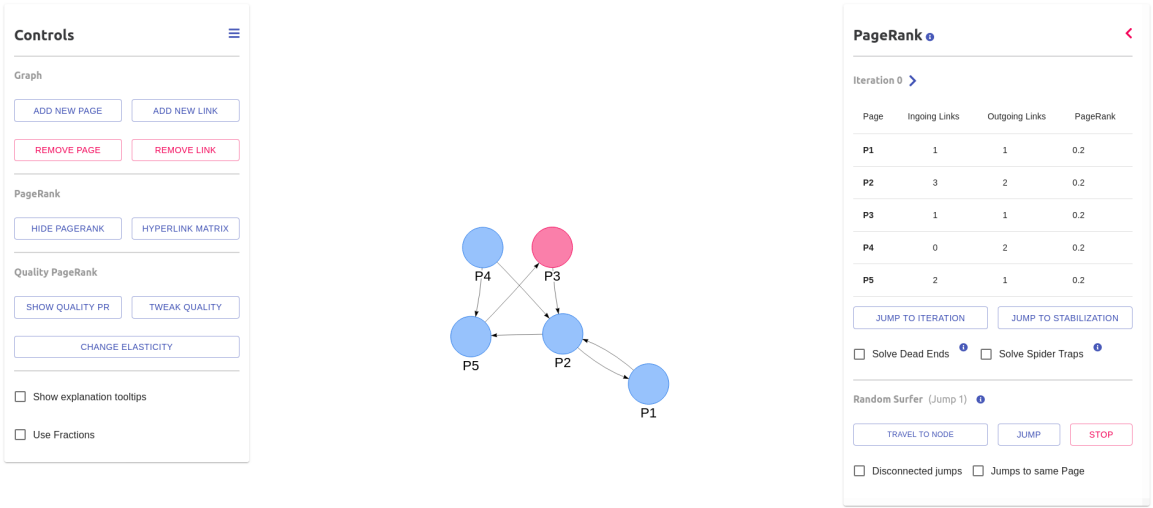


Figure 18: The Random Surfer's current position is showcased by highlighting their position in red

- **Disconnected jumps Jumps to same Page**
  These check-boxes allow a user to choose whether or not they want the surfer to make disconnected jumps or to remain on the same page, even when trying to jump. Note that these can be used to allow the surfer to attempt to solve dead end and spider trap anomalies without having to change the default HyperLink matrix.

4.3.3 Quality PageRank Controls

The Quality PageRank control set is very similar to the normal PageRank's and as such we wont go into much detail on them. Some standout changes do exist, like the table now showing the base quality, base page rank and current page rank, in contrast to the PageRank table which showed the number of in and outgoing links as well as the iteration's page rank values. For the random surfer the highlight is now green rather than red in order to diferentiate it from the normal random surfer. It's been mentioned before that when creating a new Page they have a default quality value of 10, but this can easily be modified. It should also be noted that the base values for the quality page ranks are taken from the current iteration of the PageRank algorithm.

- **SHOW/HIDE QUALITY PR**
  This button behaves exactly like the *SHOW/HIDE PAGERANK* button, except it now opens the Quality PageRank right side menu.
- **TWEAK QUALITY**
  This button opens a modal that allows a user to pick from the list of available pages in order to change their quality value. When picking a page the quality input immediately changes to that page's current quality value.
- **CHANGE ELASTICITY**
  A user can change the current elasticity value being used for the quality page rank computations.

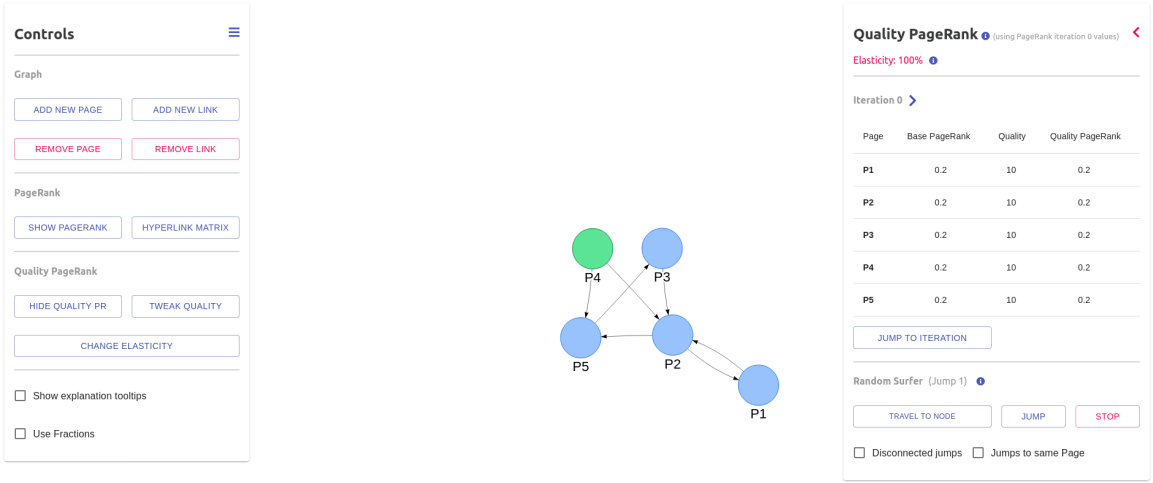


Figure 19: The Quality PageRank side menu shows what the current PageRank's iterations is and what the elasticity value is set to.

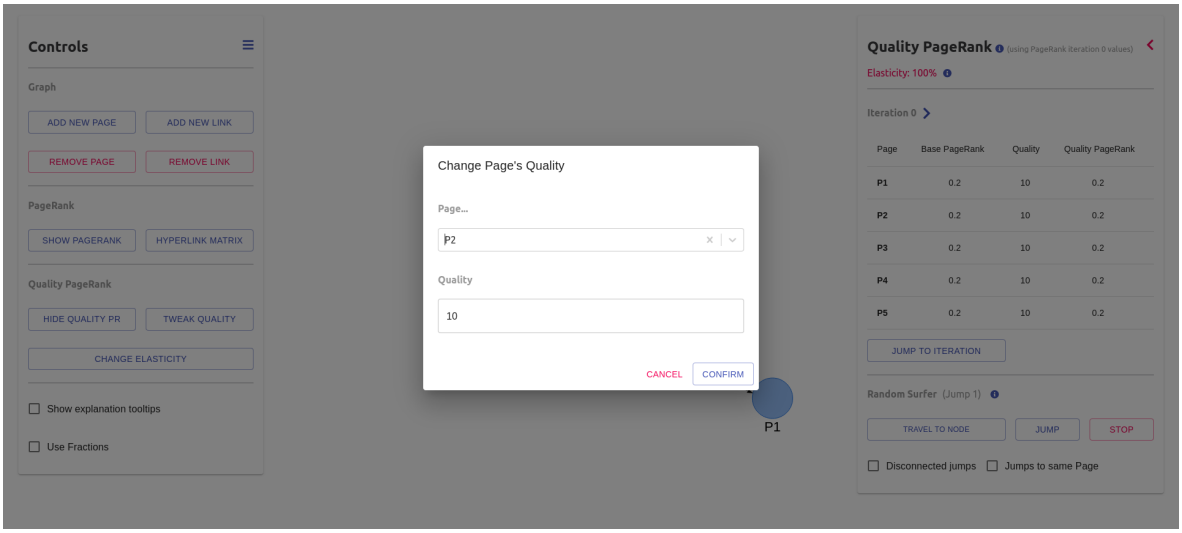


Figure 20: A user can freely change the quality of any of the pages.

4.4 Limitations and Possible Improvements

Overall all objectives for the web-app were met up to a standard of quality we are happy about. This, however, does not mean that some consolidations had to be met, and given more time some aspects would have to be improved. Mainly there's the fact that computing the algorithms past iteration 100000 is rather slow and causes the webpage to freeze slightly. Given more time improvements could be made to make this faster.

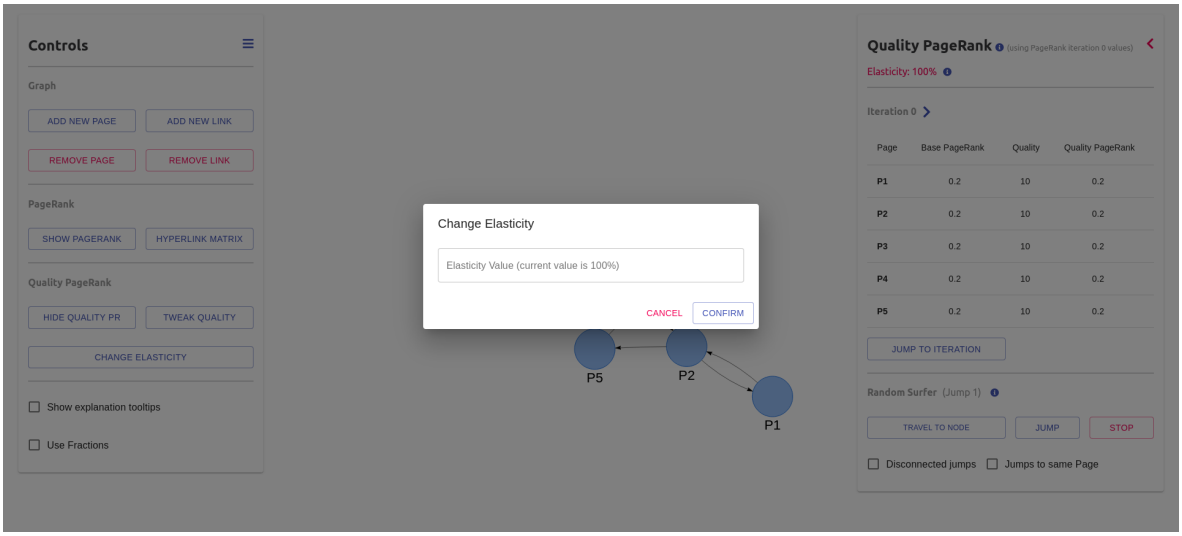


Figure 21: A user can freely change the elasticity value.

5 Conclusion

To finalize this report, and looking back at all that was made we believe to have both built an appealing and vast tool for the exploration of the PageRank concepts, as well as having deepened our knowledge on these algorithms ourselves, besides having learned more about market evolution due to the quality of the product and elasticity of the market.

6 References

[1] Diogo Silva, Vasco Ramos, João Vasconcelos, Tiago Mendes, *Search Engines: Google's Page Rank*, Not publically available [10/02/2020]
10, apr. 2020.

[2] React JS team, *React: Getting Started*,
Available Online [10/02/2020]: https://reactjs.org/docs/getting-started.html

[3] Jure Leskovec, Anand Rajaraman, Jeff Ullman, *Mining of Massive Datasets, Ch. 5 Link Analysis*,
Available Online [10/02/2020]: http://infolab.stanford.edu/~ullman/mmds/ch5.pdf
2014.

[4] António Teixeira, *Aula 23-24 - PageRank (Official class slides)*,
Available online for students of the Métodos Probabísticos para Engenharia Informática on the course's page [01/01/2019]: https://elearning.ua.pt/
2010.

[5] Manuel de Oliveira Duarte, Hugo Félix, *Exemplos de Utilização da Ferramenta Vensim*,
Available online for students of the Aspetos de Engenharia Informática e Software on the course's page [10/02/2020]: https://elearning.ua.pt/
10, mar. 2015.

[6] J. Ballin, *How to Deploy Your React App to Heroku*,
Available Online [10/02/2020]: https://medium.com/hackernoon/properly-deploy-your-react-app-to-heroku-c1a13f5f978c
13, aug. 2018.

[7] Shakhor Smith, *Deploy Your React App To Heroku*,
Available Online [10/02/2020]: https://dev.to/smithmanny/deploy-your-react-app-to-heroku-2b6f
23, mar. 2018.

[8] Vis.js Team, *Vis.js Network Documentation*,
Available Online [10/02/2020]: https://visjs.github.io/vis-network/docs/network/

[9] Fadi Khadra, *React Toastify*,
Available Online [11/02/2020]: https://fkhadra.github.io/react-toastify/

[10] Jos de Jong, *math.js*,
Available Online [11/02/2020]: https://mathjs.org/

[11] MaterialUI Team, *MaterialUI Documentation*,
Available Online [11/02/2020]: https://material-ui.com/

[12] Jed Watson, *React-Select*,
Available Online [11/02/2020]: https://react-select.com/home

7 Appendix

7.1 Template

This document was elaborated with the use of André Brandão's LaTeXTemplate, provided in the class' page.

7.2 PageRank-Visualizer URL

The PageRank-Visualizer Web-App is publicly available and hosted on Heroku on this URL.