SIPping Wikipedia

Using Statistically Improbable Phrases (SIPs) to Determine Document Relatedness

Alexandre Boulgakov
Computer and Mathematical Sciences
California Institute of Technology
Pasadena, CA 91125
boulgako@caltech.edu

Giordon Stark
Physics, Math, and Astronomy
California Institute of Technology
Pasadena, CA 91125
kratsg@caltech.edu

Abstract

Statistically improbable phrases, phrases that occur more frequently in a particular document than they do in a containing corpus, are currently used as keywords. We propose a framework that uses statistically improbable phrases to quantify relatedness between pages. We test the framework with a dataset drawn from Wikipedia, and propose potential applications.

Categories and Subject Descriptors I.2.7 [*Natural Language Processing*]: Text Analysis; H.2.8 [*Database Applications*]: Data Mining

General Terms Algorithms, Human Factors, Measurement

General Terms Statistically Improbable Phrases, Term Frequency, Inverse Document Frequency, Heavy-Tailed Distributions, tf-idf, SIPs

1. Introduction

Statistically Improbable Phrases (SIPs) were first developed by Amazon.com for their "Search Inside!" program. Amazon scans the text of all books in a particular category and determines the phrases that occur with high frequency in a particular book relative to all books in the category. These phrases are denoted SIPs. They identify interesting, distinctive, or unlikely phrases that occur in the document (with respect to a corpus) – a search ranking technique for Amazon to generate tags for a book. In a more general sense, SIPs are a set of keywords that are likely to be more relevant for search engines in generating desirable results. An example of the inverse is a "Googlewhack" in which users combine two dictionary words together to form a new phrase that returns exactly one hit on Google.

In order to determine SIPs, we can use a weighting system similar to term frequency – inverse document frequency (*tf-idf*). This provides a statistical measure to evaluate the relevance of a particular "word"[1] in a given document relative to all documents in the collection. The actual ranking of a word can be determined by multiplying the two independent rankings: term frequency and inverse document frequency. For example, suppose a set of papers on Networks talk about centrality and closeness. We can determine the term frequency of the phrase "degree of centrality" in a particular document by counting the number of times that phrase appears in the document and normalize it using the number of trigrams in that document. Then, we find the inverse document frequency by taking the number of documents in our collection and dividing by the number of documents in our collection of research papers that use this same phrase (excluding the current document we're analyzing). This ranking favors phrases found abundantly in the document (term frequency) but not the corpus (document frequency). It also has the added benefit of filtering out common words (stop words) due to their high document frequency.

[1] By word, we also imply a bigram or trigram (a phrase of two or three words respectively)

1.1 Related Work [3]

Amazon, LLC pioneered most of the work in this field. Due to the commercial liabilities with their marketplace book search - Amazon.com does not appear to elaborate on their methodology for determining and generating SIPs, however S. Anand has created a similar project in Python. The source of 100 e-books were parsed to get 100,000 words. These words were hard-coded into his script for comparison against a web page submitted for analysis. Anand considered using a regular expression to get all the words extracted out of the web page, but this posed issues because his definition of word required keeping in the punctuation marks. We will improve on this with consistency with respect to filtering and parsing. Consistent rules will give us the same results with or without punctuation marks (*don't* converts to *dont*). Our goal is to determine page-relatedness based on these shared phrases.

Anand uses an algorithm for *tf-idfs* which will produces a heavy-tailed distribution divided by a heavy-tailed distribution, which will not necessarily be heavy-tailed. We improve on this by modifying the IDF formula to produce the heavy-tailed *tf-idfs* that [6] predicts.

Next - he appears to have no clear idea of setting the threshold for which words are SIPs, but simply choosing "words occuring 10 times as often as in normal text". We will use boxplots to analyze the overall distribution of *tf-idfs* for our Wikipedia articles and find that threshold using an average of multiple datasets.

Finally, the biggest issue with the Python script is the scalability. Anand relies on calling a url and an external service to parse out the HTML (this also includes text not shown on the page) and fetch the words. Large HTML files and slow HTTP connections will slow down this script significantly that it is not applicable for numerous projects involving large data sets on the order of megabytes to gigabytes! A secondary goal with this project is to

be able to compute and compile SIPs for large data sets with a compact script with no dependencies.

2. Data Gathering

Our data was acquired from `http://dumps.wikimedia.org/` and parsed with custom-made scripts to filter out the WikiML and extract the phrases.

2.1 What is a Phrase?

Before we could start analyzing the phrases within the articles – we needed a definition of "phrase" as a single word, bigram, trigram, etc that was both meaningful and practical. In an ideal n-gram model, the n^{th}word depends only on the $n-1$ words before it. We also treated punctuation marks as delimiters so our phrases do not span sentences. Our preliminary results determined that all non-redirect, content-filled pages gave us over 1.5 billion bigrams — which are not useful for sorting short articles. Instead, we opted for single word phrases since Wikipedia articles are very short on average compared to books and novels. This would be more practical for our purposes in the long-run.

2.2 Is our Data Set Large Enough?

Wikipedia has 5.4 million articles that are non-redirects (13.2 million articles total). Of these, only 2.3 million articles (roughly 50%) had at least 100 words in their content. Given physical limitations on the computers in which we ran the scripts, we could only compute a subset of 100,000 articles at a time. We chose a random 100,000 articles which gave greater than 4.1 million links only on those articles. This accounted for 1.6% of the edges total and for 5% of the total articles.

3. Finding Statistically Improbable Phrases (SIPs)

In order to determine statistically improbable phrases, we assign a ranking to every phrase in our corpus – a *tf-idf* weight. In the first iteration of the project – our goal is to use the most simplified form of the *tf-idf* so we can generate rough results to see how viable our endgame is. Mathematically,

$$\text{tf}_{ij} = \frac{n_{ij}}{\sum_k n_{kj}} = \frac{n_{ij}}{|d_j|} \tag{1}$$

Our term frequency is the number of times a given term appears in the document divided by the number of terms in the particular document. In equation 1, n_{ij} is defined to be the number of occurrences of term t_i in document d_j. The total number of terms in document j is $|d_j| = \sum_k n_{kj}$. On its own – the term-frequency will provide us with a heavy-tailed distribution[6].

$$\text{idf}_i = \frac{\sum_k d_k}{|\{j : t_i \in d_j\}|} = \frac{|D|}{|\{j : t_i \in d_j\}|} \tag{2}$$

The inverse document frequency is the total number of documents in our corpus divided by the number of documents containing the term t_i. Our IDF happens to be "inverse" heavy-tailed so the naive approach of combining equations (1) and (2) would compute our *tf-idf* and cancel each other out – giving us meaningless data. Instead, we use a function that diminishes the heavy-tails of the idf without making it meaningless – the logarithm function.

$$\text{idf}_i = \log\left(\frac{|D|}{|\{j : t_i \in d_j\}|}\right) \tag{3}$$

Finally, by computing the *tf-idf* — the distribution retains meaning of heavy-tailed (see Figure 1).

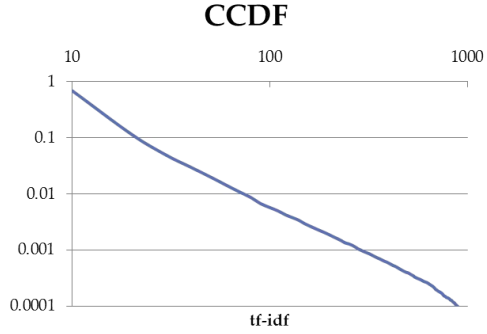

Figure 1.
Heavy-Tailed Property of *tf-idf*

4. Applying SIPs

4.1 SIPs as a Representation of Content

In existing systems such as [2], SIPs are used to assign keywords to documents. This demonstrates that the SIPs associated with a document are a summary of the document's content and the tf-idf can be viewed as the weight assigned to each word in a document. That the SIPs represent content implies that we should be able to obtain an objective measure of the relatedness between two documents by using the SIPs associated with each document.

Since the English Wikipedia has more than five million articles, there are over 25 trillion ordered pairs of articles (for some interpretations of relatedness, the comparison might be directed), making storage and calculation of a relatedness index for each pair infeasible. In addition, it is unclear why storage of such a dataset might be necessary.

4.2 Links between Wikipedia Articles

Wikipedia makes it very simple to add links between articles, and there are people who abuse this system. There are both vandals who abuse the system intentionally and inject spam links into popular pages, and those who add extraneous links without malicious intent. The latter is very common with new Wikipedia editors who think that every word on a page should link to the corresponding page. Unfortunately, this can quickly lead to pages containing so many links that it is unclear which are worth clicking. Some examples include the link from "Ford City, California" to "Pentland, California" (the two cities are not related beyond being located in the same state) and the link from "Soy milk" to "Bai ye" (the relationship responsible for the link is that bai ye is derived from tofu, which in turn is made from soy as is soy milk). The prominence of such links is evidenced by the popularity of the "Wikipedia game" wherein players pick two unrelated articles and attempt to find a link path from one to the other.

A common use case for Wikipedia is to look up articles related to a specific subject, and a common method is to find a single article related to this subject and follow links from that article. Links to unrelated articles make it difficult to navigate within a cluster of articles related to the target subject. This suggests that relatedness could be used to cluster related articles and connect them with links. Other links might be removed to avoid "link pollution" where the relevant links are obscured by irrelevant links.

4.3 SIP-wise Relatedness Index

We will first propose a general framework for using SIPs to determine relatedness of articles. Then we will present concrete implementations of the framework and analyze their performance on a large subset of Wikipedia.

4.3.1 Requirements

The framework must:

1. Rely only on the SIPs (and associated *tf-idfs*) of the two articles it is comparing. It must not use any additional information, since that might lead to complexity that scales with the size of the dataset, which is unacceptable since it would prevent its use for Wikipedia (which is already large and is growing) and for larger data sets such as the World Wide Web.
2. Not penalize articles for growing. As an invalid example, a framework that compares two articles based on the fraction of SIPs they share would penalize the articles if any one of them grows to include more SIPs (even if it is not at the expense of the existing SIPs). This is undesirable because if a fraction of an article has some content, the entire article contains that content as well.
3. Be able to handle articles of varying length. Wikipedia articles range from "stubs" that are a few words long to articles (mainly lists) that can reach hundreds of printed pages in length.

The first requirement is easy to satisfy. We will consider the second requirement later. The third requirement, however, requires some thought. A naïve solution might look at, say, the top 10 SIPs for each article and count how many they have in common. This seems reasonable at first since one might expect the top SIPs to provide a summary of each article's content. However, this would handle articles of different length differently. An article with only 10 words would have all of those words be SIPs (including stop words such as "the") and would have little chance of being related to any other article. On the other hand, large articles such as "Dictionary of chemical formulas" might contain so much content that it cannot be summarized in 10 words. One might argue that if an article cannot be summarized in 10 words that it is too broad to be strongly related to any of its content, but it should be noted that our use of the *tf-idf* already takes this into account in the *tf* factor, and we do not want to penalize an article twice for the same reason. This means that in order to satisfy the third requirement we should examine all of the SIPs in an article, rather than some fixed number. *tf-idfs*

4.3.2 A Two-Parameter Solution: $RI(f, g)$

Since we will examine all of the SIPs in each article, along with associated *tf-idfs*, the input to the algorithm will be a tuple $(d_1, d_2, \text{tf-idf}_1, \text{tf-idf}_2)$, where $\text{tf-idf}_j : d_j \to \mathbb{R}$ is a function that takes a term $i \in d_j$ and returns $\text{tf-idf}_{\text{id}_j}$. Since we cannot use SIPs that the articles do not share in a constructive way and the second requirement prevents us from using them against the relatedness index, we will only look at the SIPs the articles share, namely those contained in $d_1 \bigcap d_2$. Our proposed framework will not examine the interactions between SIPs as it is unclear how to usefully do this by using only the word to *tf-idf* mapping we are given (and the first requirement prevents us from using this).

This suggests that we can decouple an algorithm into two parts: a function $f : \mathbb{R}^2 \to \mathcal{D}$ that combines $\text{tf-idf}_1(i)$ and $\text{tf-idf}_2(i)$ into an intermediate representation and a function $g : \mathcal{D}^* \to \mathbb{R}$ that combines the results of f for each$i \in d_1 \bigcap d_2$ into a single relatedness index. In fact, this decoupling is not restrictive since f can be the identity map with $D = \mathbb{R}^2$, and g would do the bulk of the computation. We call such an algorithm for computing the relatedness index $RI(f, g)$, with $RI(f, g)(d_1, d_2, \text{tf.idf}_1, \text{tf.idf}_2) = g(f(\text{tf-idf}_1(i), \text{tf-idf}_2(i)) : i \in d_1 \bigcap d_2)$ It satisfies framework requirements 1 and 3, for reasonable parameters (ones that use all of their arguments and where g satisfies requirement 2). It is left up to g to satisfy requirement 2.

4.3.3 Values for f

We will only examine $\mathcal{D} = \mathbb{R}$ in this paper due to the fact that any intermediate results we can envision producing can be represented as real numbers. It is clear the f should be nondecreasing in both of its arguments (assuming the same is true of g, see section 4.3.4), and increasing in at least one, since otherwise more improbable phrases, phrases that are more representative of an article's content, would be less significant to the overall weight. Specific values we tried were:

- **Add**: $f(x, y) = x + y$. **Add** encompasses the notion that increasing any one of the two tf-idfs should increase the result correspondingly. It can also encompass the notion of arithmetic mean, which can be obtained by a constant scaling factor of $\frac{1}{2}$.
- **Mul**: $f(x, y) = x \times y$. **Mul** encompasses a similar notion to that of **Add**, but more extreme, with highly improbable phrases being granted a high weight even if the corresponding phrase in the second article is not as improbable.
- **Max**: $f(x, y) = \max(x, y)$. **Max** is never larger than either x or y and represents that the relatedness of two articles should be bounded by some notion of the content in their union.
- **Rms**: $f(x, y) = \sqrt{x^2 + y^2}$. **Rms** is again a representation of average.

4.3.4 Values for g

Here we again want a function nondecreasing in all of its arguments to satisfy requirement 2. The ones we tried were:

- **Sum**: $g(S) = \sum_{v \in S} v$. **Sum** encompasses the notion that each shared SIP should contribute to the result accordingly.
- **Logsum**: $g(S) = \log\left(\sum_{v \in S} v\right)$. **Logsum** is similar to **Sum**, but viewed on a logarithmic scale.
- **Sqrtavg**: $g(S) = \frac{\sum_{v \in S} v}{\sqrt{|S|}}$. **Sqrtavg** attempts to penalize articles for excessive breadth, but not so much as to prevent expansion, since the added terms will increase the numerator faster than they will the denominator. This will, however, violate requirement 2 in some edge cases where an added term is much smaller than the already existing terms.

5. Results

We tested all pairwise combinations of f and g, denoted as fg (e.g., **AddSum**) in the plots on a subset of Wikipedia. There was little difference between values used for f, and there was little difference between $g =$ Sum and $g =$ Sqrtavg, so we will include two representative samples, **AddSum** (Figure 2) and **AddLogsum** (Figure 3), here. Please see Appendix D for the rest of the images.

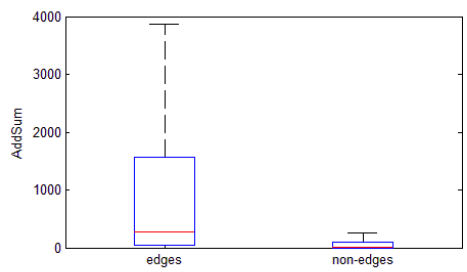

Figure 2.
Box-and-whiskers plots of the RI algorithm run on the edges within a random subset of Wikipedia and pairs of articles not connected by links — using **AddSum** method

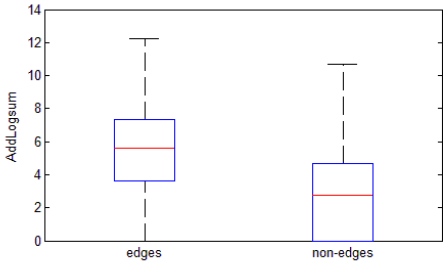

Figure 3.
Box-and-whiskers plots of the RI algorithm run on the edges within a random subset of Wikipedia and pairs of articles not connected by links — using **AddLogSum** method

It should be noted that $g = \text{Logsum}$ provides the same information as $g = \text{Sum}$, but on a different scale that make some features easier to see in each one. In Figure 2, we can see that the 75th percentile of unlinked pages ("non-edges") has **AddSum** scores way below the 50th percentile of linked pages ("edges"). However, it is unclear from the plot how the non-edges compare to the lower percentiles of the edges. This is clearer from the **AddLogSum** scores, where we see that there is indeed some overlap between the edges and nonedges. However, there are nonetheless scores that occur more frequently among the non-edges or among the edges. Specifically, estimating from the **AddLogSum** graph, more non-edges occur below 3 than edges, and more edges occur above 5.

6. Analysis

There are several interpretations for the regions of overlap. The first is that the methods used do not offer the resolution necessary to separate the edges from the non-edges. We have tried different methods which produce similar results, so we don't believe this to be the case. It might also be that it is impossible to measure relatedness using SIPs, but we don't believe this to be the case since SIPs have previously been shown to have a good representation of content and content is responsible for relatedness.

To estimate how well the relatedness index corresponds to actual relatedness, we looked at pages connected by links with the lowest relatedness indices and those with the highest relatedness indices. The results are summarized in Appendix E.

The first thing we noticed was that the highest relatedness was for a long article that linked to itself. Of course, we would expect an article to be strongly related to itself, although it is unclear why there is a link there. The next few pairs of articles were also long and about highly related subjects. The lowest RIs corresponded to articles that did not appear related.

To test in-between areas, we picked a random edge with an **AddSum** RI greater than 100. This was "Resource Base of Communist Czechoslovakia" to "Czechoslovakia", which exhibit a strong degree of relatedness, as the RI would suggest. As we were looking through the "Resource base of Communist Czechoslovakia" article, we noticed there was a link to "Salt" (the extent of the relatedness was that salt was a minor resource in Slovakia). We expected this link to have a much lower RI than the link to Czechoslovakia, and it was in fact 0.

Now, we are left to examine the final and most promising interpretation of the region of edge/non-edge overlap, that relatedness is not currently directly correlated with presence of a link. As we argued in section 4.2, we believe that this is a problem, and the data appears to suggest that this problem is present. We propose two ways to use the methods presented in this paper to make the Wikipedia link ecosystem cleaner. These ways are described with more detail in Section 7.3.

7. Future Work

7.1 Increasing the Subset

We've shown in section 2.2 that having 100,000 random articles was a good representation of the Wikipedia data set. However, we can improve our data analysis by increasing the number of articles up to the full set with the revised procedure:

- Read through the Wikipedia article text dump
- Go through the first 100,000 articles — making word frequency and inverse document frequency dictionaries
- Output these counts into a randomly-named, timestamped file.
- Continue with the next 100,000 articles until you reach the end of the file.

After processing all the articles, we'll have around 23 files, each filled with about 45 million phrases and their document & corpus frequency. We'll combine all of these files and compute *tf-idfs* using the total absolute counts. From here, we can make boxplots to determine the threshold for SIPs and be able to generate SIPs for every single document in the Wikipedia corpus.

7.2 Increasing the Number of Words in a "Phrase"

In section 2.1, the discussion of single-word phrases was prompted by the average length of a Wikipedia article. The script that has been developed to parse and weed out the phrases can handle general n-word phrases and this would be a project with interesting results. One of the expected outcomes is to generate a smaller range of *tf-idfs* and hence less SIPs — a much more manageable result with regards to marketing and searching applications.

7.3 Improving the Quality of Wikipedia Articles

There are many potential applications with having SIPs for an article and calculating their relatedness index. In this section, we describe methods in which the computer makes a decision about changes to Wikipedia articles to preserve their clarity and reduce the amount of "spammy" links. Although it is unclear whether the algorithms we present are the best that we can do, it is clear that they are a step in the right direction, and could help create a more semantic web with semantic links automatically generated from content. Below, we describe two measures that could be undertaken to facilitate users and editors alike.

7.3.1 Preventative

We can remove or prevent addition of links between pages that do not meet a minimum relatedness threshold. This only requires a single precomputed SIP database of size similar to that of the full article text dump to be stored, since the relatedness index can be computed on the fly for each request (AddSum requires only simple addition, and is I/O-bound, as are all of the other algorithms covered here). However, the SIP database requires global knowledge (to compute the idf term) and cannot be quickly computed on the fly. As far as user impact, incorrect deletion of links may be undesirable, but having a minimum requirement on relatedness for new links would be significantly more acceptable, since users would see exactly why their request is being denied, and might even rethink their decision. If they decide to continue placing a link, they would be able to add to the articles in question to increase their relatedness, in the process adding content.

7.3.2 Automative

We can go through each pair of articles that are not connected by a link (or start with an article and compare it to the articles that correspond to its highest SIPs) and if they are related above a threshold, add a link. From the graphs it appears that this would be more reliable, since the areas covered by only edges are larger than those covered by only non-edges. However, it does not fight the problem of link spamming. Instead, it increases the connectedness of Wikipedia with quality links.

References

[1] alias I. Significant phrases tutorial. 2003-2010.

[2] LLC Amazon. Amazon.com: What are statistically improbable phrases?

[3] S. Anand. Statistically improbable phrases | s-anand.net. August 2006.

[4] Ted Dunning. Accurate methods for the statistic of surprise and coincidence. 1993.

[5] Angela C. George Tara C. Long Michael A. Skinner Jonathan D. Wren Mounir Errami, Zhaohui Sun and Harold R. Garner. Identifying duplicate content using statistically improbable phrases. June 2010.

[6] Jason Rennie. The log-log term frequency distribution. July 2005.

A. Generate Unweighted Wikipedia Page Link Graph

First, we attempted to extract the structure of the Wikipedia graph from the readily available Wikipedia database dumps: a mapping from page ID to page title, a redirect list, and a link list. Wikipedia is structured such that each redirect or link entry has a page ID on the source side, but a page title on the destination side. The dumping process is not atomic so there are often inconsistencies between or even within these dumps resulting in artifacts like broken links and redirection cycles. In addition, the full-text dumps are usually incomplete. To deal with these unavoidable issues, we isolated a consistent subset of pages that appeared in all dumps. Once this set was isolated, we created an edge list representation of the directed page-link graph with each node representing a class of pages connected only by redirects, and each edge representing a link from the source node to the destination node. We chose this structure because it most closely represents what a typical Wikipedia user will see since redirect pages contain no content, and are not seen during regular browsing.

B. Extract phrases from article

Wikipedia has a set of principles – one of which is that the plain text you type should, in general, be text that gets rendered. The Wiki Markup Language (WML) will tend to be derived from standard, strict HTML and XHTML – this has limitations. By focusing less on the presentation and arrangement of text and more on the content – we lose syntax meaning and language consistency over a wide variety of pages on Wikipedia (and Mediawiki) that use WML. This causes difficulties in creating an efficient parser that will strip out all formatting, images, links, and tables; and leave behind the context on those pages. The parser within Wikipedia and built into the Mediawiki framework is not efficient for "just-in-time" computing as it preprocesses WML to create a static image of the page to be fetched by server calls. This preprocessing takes considerably more time. What follows is an example of the difficulties in parsing a few basic formatting rules defined for WML.

Basic formatting consists of bolding, italicizing, underlining, font type, and changing the size. Both bolding and italicizing involve placing a large amount of single-quotes around the text (''italic'' or '''bold''') but underlining uses standard HTML (<u>underline</u>). Displaying information also involves a lax standard of WML since there is no end to the level of nesting involved with tables[2], ordered lists, unordered lists, definition lists, and images[3]. This code could also be improperly written so the wikimedia parser implementation may differ from the standard. We also have difficulties determining what part of the text of a page is context of the idea it represents when we use

- internal anchor links, redirects, piped links[4]
  ```
  [[#See also]]
  #REDIRECT[[Main Page]]
  [[Main page|different text]]
  ```
- images[5]
  ```
  [[File:example.jpg|caption]]
  [[File:example.jpg|border|caption]]
  [[File:example.jpg|alignment|border|caption]]
  ```

or many other combinations on the page. To parse correctly – we have created a translation-replacement table, available in python by invoking the C-implementation:

```
text.translate(string.maketrans("",""), '"\'\\')
```

which allows us to rapidly remove basic formatting, lists, and single character-entities [the example shown removes single quotes and double quotes from our text which would be useful for not having to escape later]. Then, we created some regular expressions which are compiled at runtime to fix up links by translating `[[Link|Text]]` and `[[Text]]` to `Text` and also strip out all images leaving their captions behind.

```
r"\[\[(?:(?:[^|](?!\]\]))*\|)?(?:[a-z]+\:)?(.*?)\]\]"
```

Our definition of intended content on a Wikipedia article is the page itself, completely void of formatting, images, links, and other wml entities. We also need to handle tables that could nest improperly but parsing efficiently happens to be an unsolved programming problem because regular expressions are either greedy (they match as much as possible) or not-greedy (they match as little as possible).

Whenever we had code of the form `{{...}}` or `{{...{{...}}...}}`, regular expressions that try to match `{{...}}` will either be greedy and could match `{{...{{...}}` randomly or it could be not-greedy and only match `{{...}}` exactly. In the case of a greedy expression, we're left two curly braces and content after which we can't parse correctly. In the case of a not-greedy expression, we need to run through the string again to catch any nested elements. The efficient solution here is to create a non-greedy regular expression that first matches `{{...{{...}}` exactly and then continues with less-nested forms. Once we've parsed throughout the whole text leaving the punctuation and stripping out all other non-alphabet characters, we map that text into a list of n-word phrases by:

1. Splitting it into a list of sentences delimited by punctuation marks
2. Mapping a function onto each list to generate a list of phrases,
3. Flattening the list, and
4. Pushing it to a database or similar storage mechanism

```
map(lambda x: makePhrases(findall(r'\w+',x)),text)
```

[2] `http://www.mediawiki.org/wiki/Help:Tables`

[3] `http://www.mediawiki.org/wiki/Help:Formatting`

[4] `http://www.mediawiki.org/wiki/Help:Links`

[5] `http://www.mediawiki.org/wiki/Help:Images`

This produces a list of phrases from a single Wikipedia article. We're able to map 100,000 articles into about 30 million phrases in approximately 3 minutes. The overall time from reading the file, computing the phrases, and saving to a file is about 30 – 45 minutes, largely depending on the total length of all the 100,000 articles we read.

C. Example Output of *tf-idfs*

Below is a table excerpt of the output generated by our script for analyzing and determining SIPs. Please note that the output is normally tab-separated and we chose the first 30 (which are alphabetized by default).

Article ID	Word	*tf-idfs*
5322	a	0.000000
5322	abbreviated	5.393628
5322	about	3.583519
5322	abrogated	8.047190
5322	absence	4.477337
5322	accepted	3.637586
5322	accused	4.356709
5322	acreage	7.901748
5322	action	3.044522
5322	activities	3.583519
5322	acts	3.688879
5322	adam	8.496990
5322	advanced	3.761200
5322	advantage	4.094345
5322	adverse	5.713733
5322	advocate	4.976734
5322	after	11.090355
5322	aftsjeggo	11.512935
5322	against	4.394449
5322	agrarian	6.469250
5322	agreement	7.613325
5322	agricultural	4.204693
5322	agriculture	8.439015
5322	aircraft	3.850148
5322	airplane	5.789960
5322	alexander	3.737670
5322	all	1.386294
5322	allies	4.624973
5322	allocated	5.476464

D. Box-and-Whiskers Plots

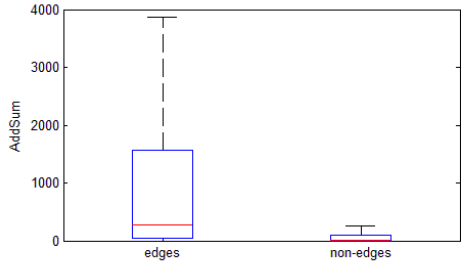

Figure 4.
AddSum

E. Table of Analysis Results

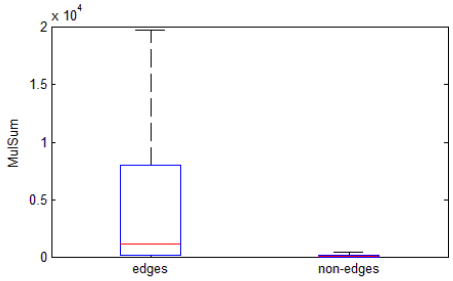

Figure 5.
MulSum

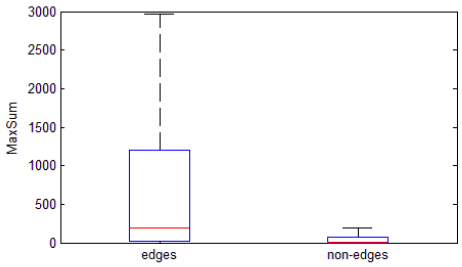

Figure 6.
MaxSum

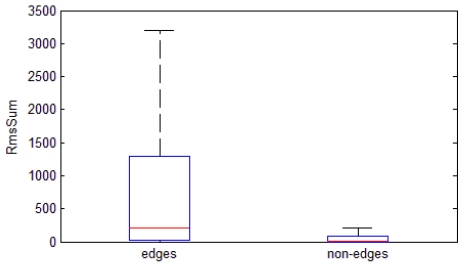

Figure 7.
RmsSum

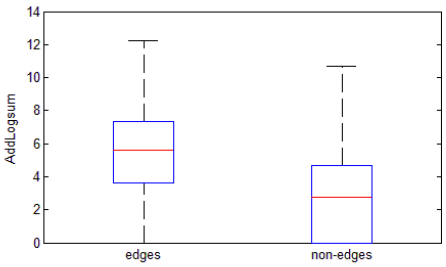

Figure 8.
AddLogSum

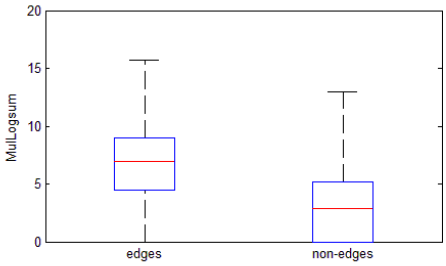

Figure 9.
MulLogSum

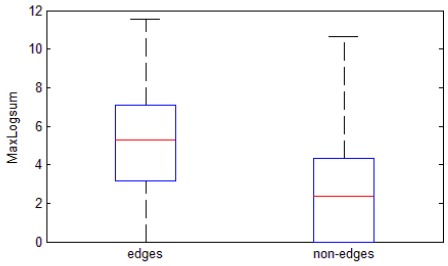

Figure 10.
MaxLogSum

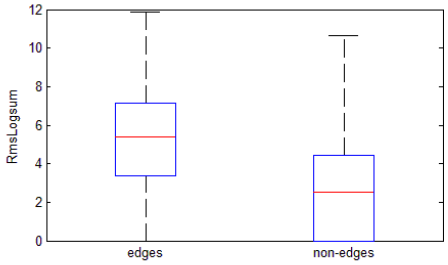

Figure 11.
RmsLogSum

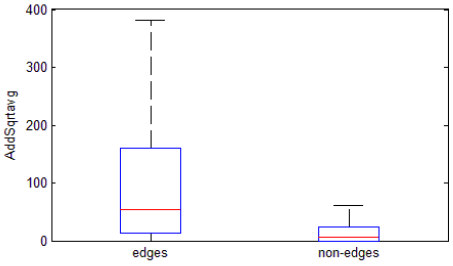

Figure 12.
AddSqrtAvg

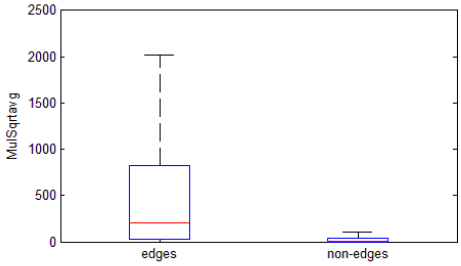

Figure 13.
MulSqrtAvg

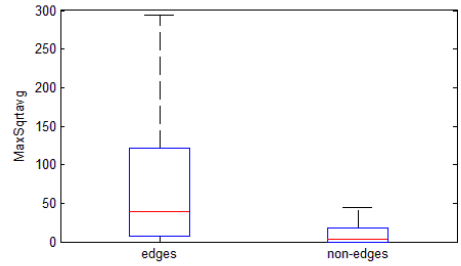

Figure 14.
MaxSqrtAvg

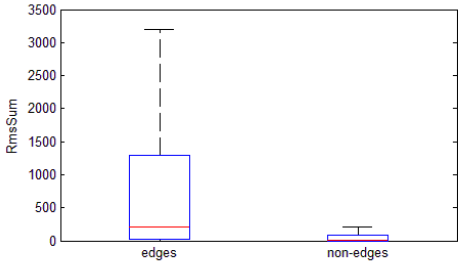

Figure 15.
RmsSqrtAvg

Source	Destination	RI	Notes
Highest RI			
List of ancient Egyptians	List of ancient Egyptians	1.1×10^8	Links to itself, highest RI
List of Acts of the Parliament of Great Britain, 1780–1800	List of Acts of the Parliament of the United Kingdom, 1840–1859	8.7×10^7	Second highest RI
Dictionary of chemical formulas	Inorganic compounds by element	3.3×10^7	Highest RI among links where the source and destination aren't nearly identical
Lowest RI			
Soy milk	Bai ye	0	
Ford City, California	Pentland, California	0.48	Lowest nonzero RI (shared with many others)
Random			
Resource base of Communist Czechoslovakia	Czechoslovakia	172.5	Random edge
Resource base of Communist Russia	Salt	0	A link from "Resource base of Communist Russia" to an unrelated page

Table 1.