Creating a Thesaurus

Nikolai Bode *University of York*
Matthew England *Heriot-Watt University*
Graham Morris *University of Oxford*
David Sibley *University of Bath*
Louise Smith *University of Strathclyde*

Problem presented, and instruction given by

Phil Knight

University of Strathclyde

Abstract

In this project we adopt the role of a lexicographer confronted with the following problem; *given a dictionary, can we build a thesaurus?* In other words, can we use the information contained in dictionary definitions to catalogue words along with their synonyms.

To do this we model the information contained in the dictionary as a network, with individual words viewed as nodes. We then build connections between the nodes using the definitions of the words. These can be weighted or directed depending on how we interpret linguistic rules.

We describe several approaches taken and how these led to our final method. We compare the final method to previous work on this problem and test the robustness of the method to noise in the original data. Finally, we develop some of the necessary tools that would be needed for the maintenance of such a thesaurus.

The 1st UK Graduate Modelling Week was organised by
The University of Nottingham

and was supported by
LMS/EPSRC

13 May 2009

Contents

1 Introduction

The Oxford English Dictionary defines a thesaurus as, *a book that lists words in groups of synonyms and related concepts*, [1]. Our problem is to automate thesaurus generation given the initial data set of a dictionary.

This specific problem falls in the wider field of data mining, the science of extracting useful information from large data sets or databases. Examples range from the experiments to understand subatomic particles at CERN to the data collected when using major supermarket reward cards. In the modern digital world it is essential to be able to extract relevant information from these vast data sets.

We start this report by giving a description of the problem. We then discuss the mathematical models we used to solve the problem in Section 2, before presenting our final solution in Section 3. Here we also give a comparison to previous work and a discussion of how robust our method is. Finally, in Section 4 we develop some tools for thesaurus maintenance before finishing with our conclusions in Section 5. This report is submitted alongside a suite of MATLAB files which contain the data and algorithms discussed. Appendix A.1 gives some information on how to use these files.

1.1 Problem description

The original dictionary data we work with comes in the form of two data files comprising the entire Webster's unabridged 1913 dictionary. The first data file gives a list of the words contained in the dictionary, 112169 in total, and labels each with a unique number. The second data file gives directed links between the words based on the dictionary definitions. There is a link from node i to node j if word j appears in the definition of word i. We demonstrate with an example.

Example: Consider the word *Thesaurus* defined in Webster's as below.

```
Thesaurus:  A treasury or storehouse; hence, a repository,
               especially of knowledge...
```

The first data file labelled *thesaurus* with word number 100813, while the words *a*, *treasury*, *or* and *storehouse* are labelled with numbers 4, 103041, 68520 and 95652 respectively. Hence the second data file contained,

```
        ⋮
100813    4
100813    103041
100813    68520
100813    95652
        ⋮
```

The data files were produced by Pierre Senellart and are available online at [2]. They were built starting from the Online Plain Text English Dictionary (OPT

2000). This was based on the "Project Gutenberg Etext of Webster's Unabridged Dictionary", in turn based on the 1913 US Webster's Unabridged Dictionary. The process is described in [3]. We record here some of the key notes from [3].

- Some of the words defined in the Webster's dictionary were multi-words. (e.g. **All Saints'**, **Surinam toad**.) These words have not been included in the data files.
- Prefixes or suffices mentioned in the head words of definitions have been excluded (e.g. **un-**, **-ous**).
- Many words have several meanings and are head words of multiple definitions. All multiple definitions of a single word have been gathered into a single definition.
- Cases of regular and semi-regular plurals (e.g. **daisies**, **albatrosses**), and regular verbs, assuming that irregular forms of nouns of verbs (e.g. **oxen**, **sought**) had entries in the dictionary have been dealt with.
- All accentuated characters were replaced in the data files with a / (e.g. **proven/al**, **cr/che**).
- There are many misspelled words in the dictionary data files as it was built by scanning the paper edition and processing it with an OCR software.

2 Mathematical models

The Oxford English Dictionary defines a synonym as, *a word or phrase that means the same as another word or phrase in the same language*, [1]. The way in which we define synonymy mathematically will be the most important factor in the success of our thesaurus generation algorithm.

2.1 Developing a definition of synonymy

To solve our problem we will develop and refine a mathematical approximation to synonymy. We start with a simple model in which two words A and B are defined to be synonyms if both word A is included in the definition of word B, and also word B is included in the definition of word A. We will write this mathematically using $A \sim B$ to indicate that A is synonymous to B.

$$\text{Definition 1:} \qquad A \sim B \quad \Longleftrightarrow \quad A \in \text{Def}_B \text{ and } B \in \text{Def}_A. \tag{1}$$

We can visualise this connection with the graph in Figure 1.

We have written a synonym finding algorithm in MATLAB to implement this definition. This algorithm provided the foundations for the more complicated algorithms to be built upon. The implemented method will be described in Section

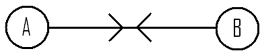


Figure 1: Schematic showing B as a synonym of A under Definition (1)

3 after we have explained the various definitions and some necessary basic graph theory ideas in Section 2.2.

Out first definition proved to be very restrictive. Unsurprisingly the requirement that both words have to be in each others definitions meant that lists of synonyms were very incomplete. Consider again our example of the word *thesaurus*. In the definition the words *treasury* and *repository* may be considered sensible synonyms. Whilst *treasury* does directly link back to thesaurus, *repository* does not.

It is possible that words will have synonyms that are not actually in the original word's definition, but that are related through other similar words. It is therefore natural to extend our first definition to include words which are related through a connecting word, or node, between them. We hence arrive at our second definition.

$$\text{Def 2:} \qquad A \sim B \iff A \in \text{Def}_B \text{ and either } \begin{array}{l} B \in \text{Def}_A \\ \text{or } \exists\, C : \; B \in \text{Def}_C \text{ and } C \in \text{Def}_A. \end{array} \tag{2}$$

This says that word A is a synonym of word B if it is a synonym under Definition (1), or if A is contained in B's definition and B is able to link back to A through a third word C. A graph representation of this second case is shown in Figure 2.

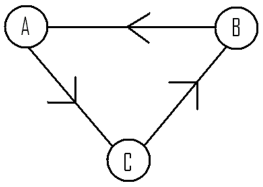


Figure 2: Schematic showing B as a synonym of A under the second case of (2)

The important thing to note about this is the cyclic nature. It is not sufficient to be able to travel from A to B; it is necessary that we can also return from B to A.

An obvious generalisation of Definition 2 is to allow an arbitrary number of steps both in and out of our original word, as represented in Figure 3. We define these cycles of n steps away and m steps back (where $n, m \in \mathbb{N}$) as **(n,m)-cycles**.

When we choose to search for synonyms as defined by (n,m)-cycles we would also search for those synonyms defined by smaller cycles. We discuss explicitly how such a definition is implemented in Section 2.3.

Upon applying this definition, we notice the balance of recall against precision. By **recall** we mean the number of correct synonyms obtained (ideally as many as possible), while **precision** refers to whether those synonyms are correct. Note that

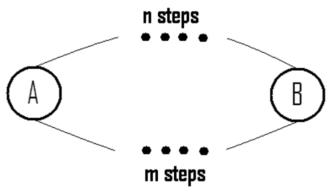


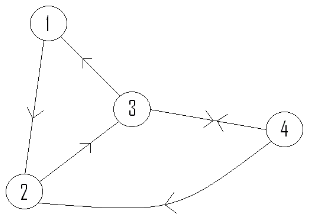
Figure 3: Schematic showing B as a synonym of A via an (n,m)-cycle

erroneous synonyms are obtained for some words even with restrictive choices of n and m such as $n = 2$, $m = 1$.

As an aside, note that the first definition was symmetric, but not necessarily transitive. The more general definition is neither except for the case when $n = m$. In all cases we do not have reflexivity since a word cannot be a synonym of itself. Therefore, for our purposes, synonymy is not an equivalence relation.

2.2 Working with the adjacency matrix

Our problem lies naturally in a graph theory setting and as such, presenting the data in an adjacency matrix proves to give us a useful way to work with the links between words. The **Adjacency matrix**, usually denoted A, is a square matrix derived from a graph. If there is an edge from node 1 to node 2, then the element $a_{1,2} = 1$, otherwise it is 0. A simple example is given in Figure 4.



$$A : \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

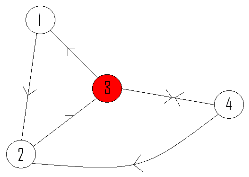
Figure 4: On the left is a simple example of a graph. The tableau on the right demonstrates how we would extract the corresponding adjacency matrix A.

We can use basic linear algebra to mimic the process of stepping along the links in the graph, and hence implement our various definitions. For example A^2 is a matrix containing information about how nodes are linked to each other in two steps. This is no longer necessarily just formed from 0's and 1's as there may now be multiple paths connecting words in two steps. This gives us a natural way to order the synonyms, as more paths connecting two words is likely to correspond to better synonymy.

Given that the adjacency matrix for our dictionary data is very large, 112169×112169, it proved impossible to compute A^2 on the machines available and so a slightly different method had to be developed. It was decided that the most obvious use of an automatic thesaurus would be a computerised, possibly online, thesaurus.

In this scenario a user would input a word and the synonyms would be calculated and displayed as they were required. This allows for far simpler computations as we are no longer building the entire thesaurus at once, in fact we are just interested in the subgraphs of A formed by stepping away from and back to the original word.

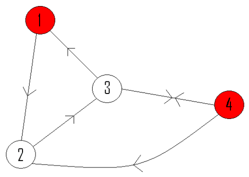
Figures 5 to 8 use the simple example introduced in Figure 4 to show how we calculate the possible paths between nodes, and hence words. Suppose we start at node 3. This can be represented by the unit vector $\mathbf{e}_3$ as shown in Figure 5.



$$\mathbf{e}_3^T = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 5: Our example graph with node 3 highlighted

We can see from the graph in Figure 6 that in one step we can move from node 3 to either node 1 or node 4. This information can be found computationally by calculating $\mathbf{e}_3^T A$.

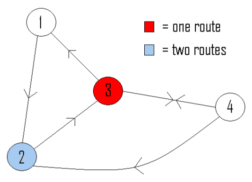


$$\mathbf{e}_3^T A = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}$$

Figure 6: Possible locations after one step away from node 3

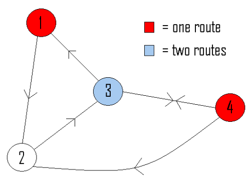
Similarly, to step twice away from node 3 we calculate $\mathbf{e}_3^T A^2$. If we were performing this for all nodes simultaneously, then A^2 would indeed need to be calculated, but as we are now stepping away from just node 3, we can do this calculation as two vector matrix multiplications $\left(\mathbf{e}_3^T A\right) A$. This requires much less memory computationally, and is necessary when working with the actual data.

Figure 7 shows that in this case there are multiple paths that allow you to move from node 3 to node 2 in two steps, (via node 1 or node 4). The weight *2* in the resulting vector will be exploited in our algorithm later as it is now natural to assume that nodes 3 and 2 are more connected, and thus more synonymous, than if they could only be connected by one path.
Finally, in Figure 8, we show the possibilities when stepping three times away from node 3. Note further that premultiplication of our word vector by the matrix A finds the possible paths back to a node. For example $A^2\mathbf{e}_3$ gives us the nodes from which we could reach node 3 in two steps.



$$\mathbf{e}_3^T A^2 = \begin{pmatrix} 0 & 2 & 1 & 0 \end{pmatrix}$$

Figure 7: Possible locations after two steps away from node 3



$$\mathbf{e}_3^T A^3 = \begin{pmatrix} 1 & 0 & 2 & 1 \end{pmatrix}$$

Figure 8: Possible locations after three steps away from node 3

2.3 Implementing the definition

Recall our general definition of synonymy given in Section 2.1. This found the synonyms of a word by looking for those words linked by **(n,m)-cycles** to the original word. Practically, the words linked by these cycles can be identified by the intersection (entries which are both non-zero) of the vectors

$$\mathbf{e}_i^T A^n \cap A^m \mathbf{e}_i. \tag{3}$$

If we chose to look for (3,1)-cycles we also include those synonyms obtained from (1,1) and (2,1)-cycles. So in general when looking for (n,m)-cycles, we in fact want the union of all possible (x, y)-cycles, where $1 \leq x \leq n$, $1 \leq y \leq m$. This means for word i we need to calculate

$$\bigcup_{(x,y)=(1,1)}^{(n,m)} \mathbf{e}_i^T A^x \cap A^y \mathbf{e}_i. \tag{4}$$

When identifying the words we take care to preserve the values in the vectors. These indicate how many different paths there were linking words and are used later to rank the synonyms.

It should be clear that the more steps away from and back to the start word, the more *synonyms* we obtain. However, it becomes apparent very quickly that many of the words are not actual synonyms. Through testing we found that the best results usually came from $(n, 1)$-cycles. We may argue that when looking up words in a dictionary, we can keep looking up connected words stepping away from the original and still obtain sensible synonyms as long as the nth word is directly connected back to the original.

Having completed a couple of trips around the modelling process we have now obtained a method which, by choosing large enough (n,m)-cycles, makes it possible to produce exhaustive lists of synonyms. We can hence obtain excellent recall, and now need to improve the precision or accuracy of our method.

2.4 Stopwords

A small improvement to our algorithm can be obtained through the removal of **stopwords**. These are the small words that appear in many definitions but do not usually give much information about the word. Examples include, *and*, *the*, *if* and *this*. As well as containing little information, these words are usually connected to many others, and so can give rise to false synonyms. Lists of stopwords are readily available, we started by using a list available at [4], and then edited and extended it for our specific use. (The list at [4] was more focused on the application to search engines, containing *words* that did not apply to our case such as *www* and *com*.) The list of stopwords we incorporated into our algorithm is given in Appendix A.2.

2.5 Weighting methods

We want to rank the synonyms we find so that we can identify those which are true synonyms, and hence improve precision. When calculating the synonyms connected by (n,m)-cycles we automatically obtain information about the number of possible paths (the numbers in the vectors of Section 2.2). However these numbers bias longer path lengths since they will naturally emit a larger number of paths. It is necessary to implement a weighting system to make sense of this information.

From the literature we found there are may ways in which you can weight nodes in a network. Three simple methods given in [5, pg 178 - 188] are

- **Degree Centrality** - number of connections to (and/or from) a node
- **Betweenness Centrality** - number of paths between other nodes travelling via the node of interest
- **Closeness Centrality** - A measure of how far away all the other nodes are in the graph by shortest distance.

Another method is **Eigenvector Centrality**, which is a measure of importance of a node in a network. It assigns relative scores to all nodes in the subgraph, and uses the idea that a node being connected to a high scoring node gives it higher importance than the same connection to a lower scoring node. Google's pagerank algorithm uses a particular variant of Eigenvector Centrality.

Eigenvector Centrality, see [6], is implemented by defining the centrality, x_i, of node i to be proportional to the average of the centralities of its network neighbours

$$x_i = \frac{1}{\lambda}\sum_{j=1}^{n} A_{ij}x_j, \tag{5}$$

with λ an arbitrary constant. In matrix notation this is then

$$\mathbf{x} = \frac{1}{\lambda}\mathbf{A}\mathbf{x}. \tag{6}$$

This is then an eigenvalue problem, and the Perron-Frobenius theorem can be used in this context to show that the principal eigenvalue will correspond to the only eigenvector which is unidirectional, [6]. The principal eigenvector corresponds to the direction of maximum change. The eigenvector centrality weighting uses the principal eigenvector of the subgraph of interest, with the ith entry corresponding to the weight of the ith word. This method has proved highly effective in ranking the importance of nodes in an entire well-connected graph, and thus gives the words which are most important to the English language. This is not useful for building our thesaurus.

A potentially more useful weighting is that of subgraph centrality, [7]. The number of closed walks of length k, i.e. the number of (n,m)-cycles in our definition, starting at node i, is given by the (i,i)-entry of A^k. We would like to give weights in decreasing order of the length of the cycle, thus giving more weight to the closest neighbours. Mathematically then, we assign the weight using the sum

$$\text{Weight of word } i = \sum_{\ell=0}^{N} c_\ell \left(A^\ell\right)_{ii}, \tag{7}$$

where N is the length of the largest closed walk we wish to consider, and where c_ℓ is a scaling of our choice (previously we had it depending on cycle length). Estrada suggests $c_\ell = 1/\ell!$, which would be the recommended scaling for a completely connected subgraph. This then can be recognised as the infinite sum of an exponential, eventually deriving

$$\text{Weight of word } i = \sum_{j=1}^{n} \left(v_j(i)\right)^2 \exp \lambda_j, \tag{8}$$

where v_j is the jth eigenvector of the subgraph adjacency matrix corresponding to the jth eigenvalue λ_j. This method of weighting has also been implemented, but due to the assumption of a very well connected subgraph (which our synonym subgraph is not), a more basic method of normalisation proves more accurate.

We found that the most effective weighting method was to simply normalise the weights given by number of possible paths in a specific (x,y)-cycle. Let $O_k(x)$ be the number of paths out from the original word, word a, to word k in x steps, and let $I_k(y)$ be the number of paths stepping back to word a from word k in y steps. We then define

$$W_k(x,y) = O_k(x) + I_k(y). \tag{9}$$

We note that the more natural calculation would be to find the total number of paths connecting the two words in a specific (x,y)-cycle by the product of $O_k(x)$ and $I_k(y)$, however when implemented the addition method proved more successful.

We then calculate the weight of each possible synonym by

$$\text{Weight of word } i = \sum_{(x,y)=(1,1)}^{(n,m)} \frac{W_i(x,y)}{\sum_{k=1}^{N} W_k(x,y)}, \tag{10}$$

where N is the total number of synonyms found from that (x,y)-cycle search. In a complete graph the number of paths of length ℓ grows like $\ell!$, and so this corresponds closely to subgraph centrality.

2.6 Complete and dense subgraphs

The final refinement to our model involves another area of graph theory concerning how dense the links between nodes are. So far we have only looked at the question of whether we can model or define synonymy between two words (which we describe as single synonyms). However, often synonyms appear in groups of more than two in which each word is synonymous to all other words (e.g. *abandon*, *leave*, *quit*). Therefore, another way to model synonymy mathematically is to search for **complete subgraphs**, (or **cliques**), in the directed dictionary graph. Complete subgraphs are subgraphs in which every word or vertex is connected to all other words in the subgraph in both directions. It seems reasonable that such subgraphs will correspond to groups of synonyms. We would expect that the recall for this method would be rather low as a lot of single synonyms might be missed out, however we would also expect results of high precision with such a strong condition.

While this sounds like a good idea in principle, there are some fundamental problems with it. Firstly, finding complete subgraphs is a graph-theoretic NP-complete problem (see [8],[9] for example). In the context of our work this means that finding complete subgraphs becomes a question of computational limitations depending on the size of the graph in question. Secondly, we found in pilot experiments that searching for complete subgraphs in our directed graph is too restrictive and consequently does not help to find synonyms at all in some cases. We therefore adapt this original idea to search instead for dense subgraphs.

For our purposes we define **dense subgraphs** as the subgraphs of our original dictionary graph where every word is connected to every other in at least one direction. To locate these dense subgraphs we can redefine the links between words to be bi-directional and then search for complete subgraphs using an existing method.

The method used to search for the complete subgraphs was the only non-bespoke code we used. The routine, called Maximal Cliques and written by Ahmad Humyn takes an undirected and unweighted graph (in the form of an upper rectangular square matrix of 1's and 0's) and returns a list of all complete subgraphs. The routine is available at the MATLAB file exchange [10].

3 Final method and comparison to previous work

We now present our final algorithm and explain briefly why this particular method was implemented. This algorithm was developed from the sequential application and refinement of the models for synonymy introduced above. We also compare the results obtained using our algorithm to previous work on the topic and investigate its robustness. The MATLAB files necessary to run this algorithm were submitted alongside this report. See Appendix A.1 for details on how to use them.

3.1 The final algorithm

A flow-chart of the final algorithm can be found in Figure 9. It would be possible to run the algorithm for all words to create a complete thesaurus. However, we

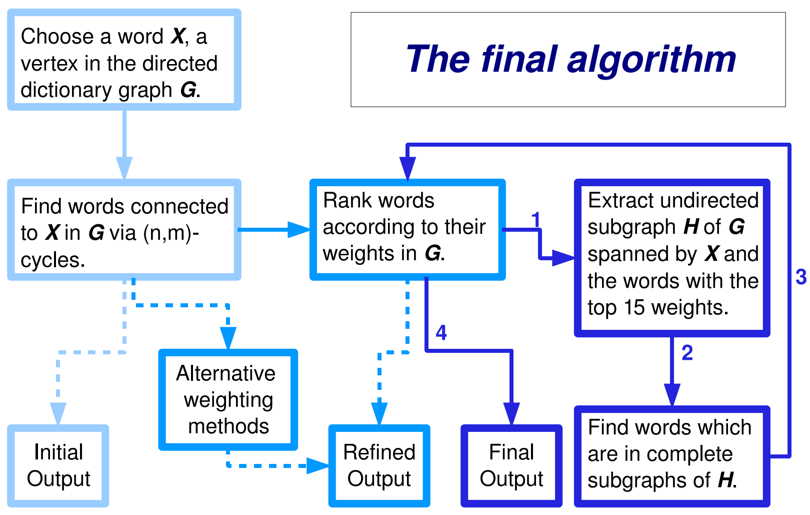

The final algorithm
Choose a word X, a vertex in the directed dictionary graph G.
Find words connected to X in G via (n,m)-cycles.
Rank words according to their weights in G.
Extract undirected subgraph H of G spanned by X and the words with the top 15 weights.
Find words which are in complete subgraphs of H.
Alternative weighting methods
Initial Output
Refined Output
Final Output
1
2
3
4

Figure 9: This flow-chart demonstrates both our final algorithm and the various iterations that were used to refine the model, (shown in progressively darker shades of blue). The final method follows the solid lines in the flow-chart, and uses the normalised weighting method from equation (10). The alternative weighting methods mainly included the ideas from Section 2.5 trying the eigenvector methods. See Section 3.1 for a more detailed description of the final algorithm.

envisaged a more interactive approach which allows the user to type in the word they want to look up (word X), and obtain suggested synonyms for this particular word. As well as being more computationally efficient this should make it easier to update the thesaurus over time.

Initially the concept of (n,m)-cycles discussed in Section 2.3 is used to investigate connections between X and other words. This method allows us to obtain a smaller subgraph of interest which reduces computation times at a good recall rate. While this initial method (light blue in Figure 9) gave good recall and some encouraging results it became clear that the precision of the results should be improved. Therefore various methods of weighting the words according to the perceived strength of their synonymy were implemented (see Section 2.5). This is shown in darker blue in Figure 9. The weighting based on normalised weights of each (i, j)-cycle from equation (10) was preferred and used in further model refinements. This method is described as the *intermediate thesaurus* in later sections.

Finally, to improve the accuracy of the results even more, the complete and dense subgraph ideas in Section 2.6 were implemented. In the interest of short computation times this approach (dark blue in Figure 9) was only applied to the subgraph H' of the dictionary graph G. This subgraph H' is spanned by X (entered by the user) and the words generated with the intermediate thesaurus with the top 15 weights.

It quickly became clear that looking for complete subgraphs within the directed graph framework gave a very low recall. We therefore converted H' into an undirected graph H by keeping all the edges but removing their directions, effectively making every edge bi-directional. We then searched for complete subgraphs in this undirected graph using the Maximal Cliques routine, [10].

All words or vertices contained in complete subgraphs of size greater than two were recorded. Finally the words found by this procedure were ranked according to their original weights (in the directed framework), so that the final set of words were ranked by their connection to the original word.

3.2 Comparison to previous work

While it is beyond the scope of this report to produce a comprehensive literature review on the topic, it is interesting to compare our methods and some of our results to previous work on the problem of extracting synonyms from a dictionary. We will consider the work of Jannink [11] and Blondel et al. [12] since they have also been applied to Webster's 1913 dictionary. We will start by introducing these alternative methods and subsequently compare the results on some test words.

Both the work of Jannink and Blondel et al. are inspired by web search algorithms which in turn use ideas from graph theory. Jannink's approach is essentially an adaptation of the pagerank algorithm which is used in the Google search engine [13]. Rather than ranking the vertices of their graph (as in pagerank), the authors suggest a relative measure between the connections of vertices. More specifically, they measure the amount of rank that flows across the connections at each iteration of their adapted pagerank algorithm which is applied to the vertices. This method of ranking connections (dubbed **ArkRank**) is then used to produce lists of related words. However, Blondel et al. remarked that this method is good at finding related words but not designed to find synonyms [12].

The method of Blondel et al., based on a generalisation of Kleinberg's web search algorithm [14], computes similarity measures between vertices. In brief, information contained in the *neighbourhood graph* of a word (words connected to the word of interest via one edge in either direction) is used to suggest synonyms. The words obtained in this way are then ranked according to a similarity measure, obtained using an eigenvector approach on the adjacency matrix of the neighbourhood graph [12]. The authors note that an obvious extension to their work would be the inclusion of more words than the ones contained in the neighbourhood graph but to our knowledge they have not yet proceeded to do so.

In [12], Blondel et al. used four words (*Disappear*, *Parallelogram*, *Science*, and *Sugar*) as examples to compare their method to other sources of synonyms. In Appendix A.3 we present our synonyms for these words alongside theirs. To quantify the quality of their results, Blondel et al. asked 21 individuals to give their suggested synonyms marks between 0 and 10, [12]. We did not repeat this somewhat subjective comparison and leave it to the reader to decide which approach works better, if any.

Since it is difficult to compare the quality of our results to results obtained previously, it only remains to assess in how far our method adds something new to what already existed. While our approach relies on the representation of the dictionary as a graph as before, we do not use ideas from web search algorithms. Furthermore, we include more words in our analysis than Blondel et al. did in their neighbourhood graphs. We tested different ways to include a relative measure between vertex connections ((n,m)-cycles in our case) and found that a simple concept based on paths between words produces the most convincing results. Finally, we investigated dense subgraphs in the dictionary graph, a conceptually simple approach with promising results. We therefore explicitly introduced the concept of groups of synonyms rather than just synonyms for individual words.

3.3 Robustness of final method

We have conducted some basic tests on a selection of words to see how robust our thesaurus was to noise in the original data. You could generate something similar to our dictionary graph by connecting 112000 nodes with 1400000 edges chosen at random. However, it would not be possible to extract semantic information from this random graph. There is a hidden structure present in the dictionary graph, which we show in this section to be robust in the face of perturbation.

First we considered the effect of typos or misspelling by changing a number of the defining links out of a word, and observing the effect this had on our thesaurus search. The target word and number of entries changed was user specified, while the entries to be changed and new words were chosen with the Matlab `rand` command.

We considered the effect of both defining words being changed into other words, and the addition of other random words into the definition. On the test cases these changes seemed to have little effect on the thesaurus results. This can be explained by the fact that it is highly unlikely the new links would direct to a word that could be linked back in a small number of steps. We note that in a case where the definition is very small with only one or two keywords, then the effect of a typo would be much greater.

Secondly we considered the effect of incorrect synonyms being entered (i.e. our adjacency matrix A contains 1 in both A_{ij} and A_{ji} when words i and j are not synonyms). Such a mistake may result from human error over the meaning of a word. Such a perturbation in the data did have a large effect on our intermediate thesaurus, but not our final version. This is because the two way links being created could give the incorrect synonyms a high weight, but is unlikely to create dense subgraphs, and so will not be picked up by the final method. Appendix A.4 contains MATLAB output to demonstrate this.

These tests lead us to the tentative conclusion that the final thesaurus is fairly robust to perturbations in the data. We use the word tentatively as it would be ideal to make the tests more thorough by increasing the number of words tested, increasing the number of tests on each word and using a better random number generator.

4 Thesaurus maintenance tools

In this section we discuss a number of tools that were developed to aid the maintenance of the thesaurus. These tools may not only be used to correct mistakes or typos present in the data, but also allow the thesaurus to be adapted to the changing nature of the English language. This second use has been highlighted since the data we are working with is from 1913, resulting in many entries that are not relevent for the modern world. They were created with the primary application of a computerised online thesaurus in mind, and as such try to be simple and user-friendly.

These tools all work on the principle of changing entries in the adjacency matrix, based on user input regarding a target word. They were created to work with the intermediate thesaurus we derived that included the final weighting system, but did not make use of complete subgraphs. It is with this thesaurus that the tools below are discussed and demonstrated. (See Section 4.3 for a note on compatibility with the final method.)

4.1 Adding new words into the system

Over time new words enter the English language which were not present before. We have developed a program that allows the user to add a new word into the system (creating a new row and column in the matrix) and then suggest some synonyms for it (put 1s in the relevent matrix entries). The existing thesaurus code can then be run with this new matrix to find additional synonyms.

For example, consider the word *bonkers* which is not present in our dictionary, but which is defined in a modern dictionary as *crazy* or *insane*. The Matlab code below shows how we can add this new word into the system and suggest these two synonyms for it.

```
>> New_Word
What is your new word? bonkers
Which A-matrix are we working with? Asw
We create a new matrix (A_new), and a new vector (index_new).
Please suggest some synonyms: crazy, insane.
>>
```

We now demonstare how the new word can be entered into the thesaurus giving a wider list of relevent synonyms.

```
>> Thes_nm_wt('bonkers ', index_new, A_new, 2,3)
Elapsed time is 0.321793 seconds.
ordsym =
    'insane'
    'crazy'
    'deranged'
    'bonkers'
    'mad'
    'insanity'
    'delirious'
    'demented'
    'derange'
    'crack'
    'distract'
    'wild'
>>
```

4.2 Changing existing words in the system

We have two similar tools that can change existing words in the system. The first is designed for circumstances where the existing definition of a word is still valid, but for which there is now an alternative meaning. For example, the word *cool* while still meaning *cold* can now also be used to mean *good*. In this case the program adds extra links between the word and some user suggested synonyms, allowing the thesaurus search to pick up a wider list.

The second tool is designed for circumstances where the existing definition is no longer valid, and the word now means something different. For example the word *terrific* was defined in 1913 as *dreadful*, or *causing terror*, and so a thesaurus search for this word provides similar results. (See Appendix A.5 for the MATLAB output relating to this example.) This use of the word is now obsolete, and in modern usage the word tends to mean the opposite. The MATLAB output in Appendix A.5 demonstrates how the user can redefine this word by suggesting a couple of more appropriate synonyms. The code severs the existing links and creates new ones. A thesaurus search then produces a wider list of relevant synonyms.

4.3 Compatibility with the final method

These tools had some compatibility issues with the *final method* discussed in the previous section. These arise because synonyms from the final method must be part of a dense subgraph as defined in Section 2.6. This was essentially a complete subgraph in the case where all edges are assumed bi-directional. The new links created by the tools above will not result in a complete subgraph, see Figure 10. Hence the target word will be missed out by the complete subgraph search.

Given more time it should be possible to modify the final method to also search for almost complete subgraphs, and thus both overcome this problem and potentially improve recall. This could possibly be achieved by allowing words that lie in $(100 - \epsilon)\%$ complete subgraphs to be allowed as synonyms.

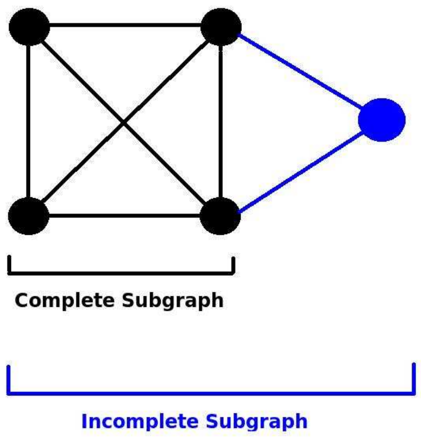


Figure 10: An example of how linking a new nodes to a complete subgraph does not result in a complete subgraph.

5 Conclusions and further work

In this report we have shown how we have created and refined mathematical models for synonymy between two words and groups of words using definitions of words in dictionaries. We have presented an application of our methods to Webster's 1913 dictionary and considered issues such as the robustness of our methods to errors in the data and the maintenance of a thesaurus in the face of a dynamic, changing language. While our concepts build on similar foundations as previous work by representing a dictionary as a graph, we have introduced new original concepts such as groups of synonyms and a novel method for weighting connections between words. These approaches produced promising results. Furthermore, we explored different types of connections between words which we called (n,m)-cycles and established that (n,1)-cycles appear to be linked more strongly to synonymy.

Despite these achievements certain topics would certainly benefit from further investigation. Webster's 1913 dictionary includes old-fashioned definitions for words which makes it difficult to compare our results to modern theasuri as definitions for many words have changed over time. Therefore, it would be interesting to apply our methods to a more up-to-date dictionary.

Dictionaries naturally comprise large data sets which imposes computational restraints on our methods. We have managed to overcome these problems in most cases by limiting our synonym search to one word at a time. However, in the search for dense subgraphs, more computationally efficient algorithms would help to increase the amount of data taken into consideration and may therefore benefit the recall rate.

Further work is also needed to deal with antonyms (words with opposite meaning). Words are ususally highly linked to their antonyms, as dictionary definitions often include words with the opposite meaning and so antonyms are often picked up when using our algorithm.

So far we have not used any linguistic information, such as word type, or relative positioning of words in the definitions. This could help to improve the accuracy of our results.

It would be interesting to apply our methods to dictionaries in other languages as has been proposed in [12]. We believe that this may help to find and understand differences in the structure of different languages and would provide a starting point to compare languages mathematically. This in turn could be beneficial for the development of automated translation protocols.

Finally, we note that previous approaches to the present problem have taken inspiration from web search algorithms (see above) which suggests strong links between the two topics. Since we have not been guided by ideas in this field it seems natural to suggest an application of our methods to web searches.

6 Acknowledgements

The authors wish to thank the organisers of the 1st UK Graduate Modelling Week. It was both very educational, enabling a clearer understanding of how to tackle applied problems, and also thoroughly enjoyable. Particular thanks to Phil Knight for bringing a really interesting problem and for the instruction during the week.

Bibliography

[1] Compact Oxford English Dictionary of Current English, Third Edition, *OUP* (2005)

[2] P. Senellart. Graph of Webster 1913, `http://pierre.senellart.com/travaux/stage_maitrise/graphe/`

[3] P. Senellart. Masters Internship Report: Extraction of information in large graphs; Automatic search for synonyms, `http://pierre.senellart.com/publications/senellart2001extraction.pdf`

[4] Ranks.NL English Stopwords, `http://www.ranks.nl/resources/stopwords.html`

[5] S. Wasserman, K. Furst, Social Network Analysis, Structural Analysis in the Social Sciences, *Cambridge University Press* (1994)

[6] M. E. J. Newman, Mathematics of networks, The New Palgrave Encyclopedia of Economics, 2nd edition, *Palgrave Macmillan* (2008).

[7] E. Estrada, J. A. Rodrguez-Velzquez, Subgraph centrality in complex networks, *Phys. Rev. E*, 71 (2005)

[8] N. Alon, Finding and counting given length cycles, *Algorithmica*, 17:209–223 (1997)

[9] R. M. Karp, Reductibility among combinatorial problems, *Univ. of California* (1972)

[10] A. Humyn, Maximal Cliques, `http://www.mathworks.com/matlabcentral/fileexchange/19889`

[11] J. Jannink, G. Wiederhold, Thesaurus entry extraction from an on-line dictionary, *Proceedings of Fusion*, 99 (1999)

[12] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, P. V. Dooren, A measure of similarity between graph vertices: applications to synonym extraction and web searching, *SIAM Review*, 46:647–666 (2004)

[13] S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine, *Computer networks and ISDN systems*, 30:107–117 (1998)

[14] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM*, 46:604–632 (1999)

Latex format based on the ESGI68 report, Reaction-diffusion models of decontamination, Dr. D Allwright.

A Appendix

A.1 Information on the MATLAB files provided

Attached to this report you should find a suite of MATLAB files relating to our problem. This appendix will discuss how they can be implemented and used.

The two data files we were given are labelled `dico` and `index`. Start by reading these into MATLAB using the `Import Data` tool. Similarly, import the text file `stopwords2` which contains the list of stopwords we use (Section 2.4). Next, run the script m-file labelled `Create_Matrix`. This will create the matrix `A` discussed in Section 2.2 and the matrix `Asw` which has had the edges attached to stopwords severed.

First we introduce three basic function m-files that we use to work with the data. The file `wordnum` gives the number by which a target word is identified. Note that the target word is written in quotes with a space at the end.

```
>> wordnum('thesaurus ',index)
ans =
      100813
>>
```

The files `definition` and `citation` can be used to find the words used in the definition of the target word, or the definitions that use that target word.

```
>> definition('thesaurus ', index,Asw)
ans =
    'applied'
    'comprehensive'
    'cyclopedia'
    'dictionary'
    'especially'
    'hence'
    'knowledge'
    'like'
    'often'
    'repository'
    'storehouse'
    'treasury'
    'work'
>> citation('thesaurus ',index,Asw)
ans =
    'thesauri'
    'treasury'
>>
```

The intermediate thesaurus which uses weighting but not dense subgraphs is run using the file `Thes_nm_wt`. The arguments are the target word, the index vector, the A-matrix and then two integers. These integers represent the number of steps in and the number of steps out (respectively) that we allow in our searches, (see Section 2.1).

```
>> Thes_nm_wt('abandon ',index,A,1,4)
Elapsed time is 0.423310 seconds.
ordsym =
    'forsake'
    'relinquish'
    'desert'
    'abandonment'
    'leave'
    'let'
    'depart'
    'resign'
    'destitute'
    'abandoned'
    'abandoning'
    'betray'
    'bury'
    'persevere'
    'deserter'
    'waive'
    'abjure'
    'reprobate'
    'revolt'
>>
```

The final algorithm (as discussed in Section 3) is implemented with the function m-file `Thesaurus`. The arguments are the same as above, but now we search for dense subgraphs. This m-files uses the subroutine `maximalCliques.m` which was obtained from [10].

```
>> Thesaurus('shout ',index,Asw,1,4)
Elapsed time is 0.537837 seconds.
ans =
    'cry'              [0.1503]
    'clamor'           [0.1503]
    'vociferate'       [0.0922]
    'exclaim'          [0.0916]
    'bawl'             [0.0916]
    'acclamation'      [0.0916]
    'shouting'         [0.0916]
    'hollow'           [0.0609]
    'cheer'            [0.0607]
    'hoot'             [0.0606]
    'huzza'            [0.0606]
    'shout'            [     0]
>>
```

Also contained are the files `Pertubate_Definition.m` and `Pertubate_Synonyms.m` which were used in Section 3.3 to check the robustness of our final algorithm. An example of their use is given in Appendix A.4. Finally, we include the files `New_Word.m`, `Replace_Word.m` and `Add_Synonym.m` for use in thesaurus maintenance and discussed in Section 4. These are simple script m-files that request user input. Explicit examples are given in Section 4.1 and Appendix A.5.

A.2 List of stopwords

The list of stopwords we use in our algorithm is given below.

```
a, about, all, also, am, an, and, any, are, as, at, be, been, but,
by, each, else, en, for, from, how, i, if, in, is, it, la, me, my,
no, of, on, or, so, than, that, the, then, there, these, this, to,
was, what, when, where, who, will, with, a, b, c, d, e, f, g, h, i,
j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.
```

A.3 Comparison with Blondel et. al

The tables below compare the results obtained with our final algorithm, using (14,1)-cycles, to the results presented by Blondel et. al. for the four test words *Disappear*, *Parallelogram*, *Science* and *Sugar*.

Disappear		**Parallelogram**	
Our results	Blondel et. al.	Our results	Blondel et. al.
Pass	Vanish	Square	Square
Vanish	Pass	Cylinder	Rhomb
Die	Die	Altitude	Parallel
Fade	Wear	Gnomon	Figure
Eat	Faint	Prism	Prism
Wear	Fade	Tetragonal	Equal
Faint	Sail	Rhomb	Opposite
Efface	Light	Rhomboid	Angles
Dissipate	Dissipate	Parallelopiped	Quadrilateral

Science		**Sugar**	
Our results	Blondel et. al.	Our results	Blondel et. al.
Law	Art	Sweet	Cane
Physical	Brance	Cane	Starch
Natural	Law	Beet	Sucrose
Art	Study	Maple	Milk
Study	Practice	Sorghum	Sweet
Skill	Natural	Sucrose	Dextrose
Chemistry	Knowledge	Piece	Molasses
Astronomy	Learning	Face	Juice
Term	Theory	Beat	Glucose

A.4 MATLAB output to check robustness of final method

In this Appendix we give some MATLAB output to demonstrate our discussion in Section 3.3. We use the word *abandon* for the example.

```
>> PertA=Pertubate_Synonyms('abandon ', 5, index, Asw);
```

Here `Asw` is the standard matrix (with stopwords removed) while `PertA` is created by adding five incorrect synonyms.

First we show the large effect this has on the intermediate thesaurus. Below we give the results with both the original and perturbed matrix.

```
>> Thes_nm_wt('abandon ',index,Asw,1,4)
Elapsed time is 0.427733 seconds.
ordsym =
    'forsake'
    'relinquish'
    'desert'
    'abandonment'
    'leave'
    'depart'
    'resign'
    'let'
    'destitute'
    'abandoned'
    'abandoning'
    'betray'
    'bury'
    'persevere'
    'deserter'
    'abjure'
    'waive'
    'revolt'
    'reprobate'
>> Thes_nm_wt('abandon ',index,PertA,1,4)
Elapsed time is 0.454440 seconds.
ordsym =
    'forsake'
    'relinquish'
    'desert'
    'abandonment'
    'consigned'
    'scape-wheel'
    'gawby'
    'oviposition'
    'postmarking'
    'leave'
    'depart'
    'resign'
    'let'
    'destitute'
    'abandoned'
    'abandoning'
    'betray'
    'bury'
    'persevere'
    'deserter'
    'abjure'
    'waive'
    'revolt'
    'reprobate'
```

Notice that several incorrect synonyms have entered the list towards the top. We now show the effect of the same perturbation on our final thesaurus.

```
>> Thesaurus('abandon ',index,Asw,1,4)
Elapsed time is 0.419323 seconds.
ans =
    'forsake'         [0.1839]
    'relinquish'      [0.1837]
    'desert'          [0.1836]
    'abandonment'     [0.1832]
    'leave'           [0.1273]
    'depart'          [0.1264]
    'resign'          [0.1262]
    'let'             [0.1259]
    'destitute'       [0.1256]
    'abandoned'       [0.1254]
    'abandoning'      [0.1254]
    'deserter'        [0.0833]
    'abandon'         [     0]
>> Thesaurus('abandon ',index,PertA,1,4)
Elapsed time is 0.478424 seconds.
ans =
    'forsake'         [0.1540]
    'relinquish'      [0.1538]
    'desert'          [0.1537]
    'abandonment'     [0.1532]
    'leave'           [0.1058]
    'depart'          [0.1049]
    'resign'          [0.1046]
    'let'             [0.1043]
    'destitute'       [0.1040]
    'abandoned'      [0.1038]
    'abandon'        [     0]
>>
```

In this case there was no decline in precision and only a small drop in recall.

A.5 MATLAB output to demonstrate the replace word tool

In this Appendix we give some MATLAB output to demonstrate the tool designed for replacing existing definitions that are no longer valid. This was discussed in Section 4.2.

The example word is *terrific* which was defined in 1913 as *dreadful*, or *causing terror*. We see that a thesaurus search with this word provides similar results.

```
>> Thes_nm_wt('terrific ',index,Asw,2,2)
Elapsed time is 0.239452 seconds.
ordsym =
    'dread'
    'terror'
    'fear'
    'terrible'
    'awe'
    'dreadful'
    'doubt'
    'affright'
    'terrific'
    'formidable'
    'strike'
    'stem'
    'apprehensive'
    'horror'
>>
```

The Matlab code below shows how we redefine this word by suggesting a couple of more appropriate synonyms. The code will severe the existing links and create new ones.

```
>> Replace_Word
Which word would you like to redefine? terrific
Which A-matrix are we altering? Asw
Your new matrix will be stored as A_Rep.
Please suggest some synonyms: splendid, marvelous
>>
```

We now see that the Thesaurus search can produce a wider list of synonyms.

```
>> Thes_nm_wt('terrific ',index,A_Rep,2,3)
```

```
Elapsed time is 0.356919 seconds.
ordsym =
    'splendid'
    'marvelous'
    'terrific'
    'brilliant'
    'illustrious'
    'magnificent'
    'shining'
    'incredible'
    'sumptuous'
    'admirable'
    'gay'
    'glorious'
    'grand'
    'legend'
>>
```