# Quantifying Substitutability

by

David X. Wang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2014

# Abstract

In this thesis, we will tackle the problem of how keyphrase extraction systems can be evaluated to reveal their true efficacy. The aim is to develop a new semantically-oriented approximate string matching criteria, one that is comparable to human judgements, but without the cost and energy associated with manual evaluation. This matching criteria can also be adapted for any information retrieval (IR) system where the evaluation process involves comparing candidate strings (produced by the IR system) to a gold standard (created by humans). Our contributions are three-fold. First, we define a new semantic relationship called substitutability – *how suitable a phrase is when used in place of another* – and then design a generic system which measures/quantifies this relationship by exploiting the interlinking structure of external knowledge sources. Second, we develop two concrete substitutability systems based on our generic design: WordSub, which is backed by WordNet; and WikiSub, which is backed by Wikipedia. Third, we construct a dataset, with the help of human volunteers, that isolates the task of measuring substitutability. This dataset is then used to evaluate the performance of our substitutability systems, along with existing approximate string matching techniques, by comparing them using a set of agreement metrics. Our results clearly demonstrate that WordSub and WikiSub comfortably outperform current approaches to approximate string matching, including both lexical-based methods, such as R-precision; and semantically-oriented techniques, such as METEOR. In fact, WikiSub's performance comes sensibly close to that of an average human volunteer, when comparing it to the optimistic (best-case) inter-human agreement.

# Acknowledgements

The success of my Master's thesis required the help of many individuals. Without them, I would not have been able the meet the deadlines and objectives required to complete my study. My deepest gratitude goes out to the following people, for their invaluable help and support:

To our Father in heaven, for creating and making me into the person I am today; for giving me strength, wisdom and the curiosity to explore the very fringes of human knowledge. Without Him, none of this would have been possible.

To my loving parents, George and Lisa, for existing as the perpetual anchors of my life that I can always rely on in my darkest hours; for providing the tough love I needed in the final drive to the finish line.

To my supervisors, Dr. Xiaoying Gao and Dr. Peter Andreae, for lending their continued guidance to keep my research on track; for the inspiration I obtained by tapping into their shared well of knowledge; and for bestowing me with the belief required to endure this long journey.

To Craig Watterson, for providing pastoral support when I was on the verge of abandoning the monumental task that laid before me; for showing me how to cope with the demotivating task of writing day after day. This entire work would have remained hidden in my own mind, till this day, without his aid and advice.

To my fellow colleagues, Roma and Sam, for being a friendly and familiar face in the office; your distractions were always a welcomed break from work.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Over the years, there have been many advances in the field of information retrieval (IR), but little has changed in the way we evaluate systems in this field. IR systems perform the task of finding useful information from a large collection. The generic approach is to compute a numeric score for each retrieved item/entity, which represents its usefulness/relevance according to some metric (e.g. a search query). These items could range from textual phrases and documents, to multimedia sources such as images, videos and audio. Once a score has been attached to each item, they are ranked according to this value. The final output of IR systems is a portion of the top ranking items, which are then presented to the user [65].

To analyse the performance of an IR system, the quality of the retrieved items are evaluated on their relevancy towards the task at hand. This judgement can be made by domain experts, but this is a time-consuming and expensive process [1]. An alternative is to integrate the IR system into an application and then indirectly evaluate its performance based on how it influences the application's efficacy. However, this introduces more parameters into the mix that are not part of the IR system itself. Therefore, experiments are difficult to control and results may become biased in unpredictable ways [73].

As researchers strive to obtain consistent and timely results, both of the previous approaches are undesirable. Instead, the preferred method comes in the form of automatic evaluation. For this to work, researchers must first create a labelled testing dataset, classifying each item as either relevant or irrelevant. While this first step can be time-consuming, all subsequent tests can be performed automatically by cross-referencing the output of the IR system against the labelled dataset (the gold standard). An automatic evaluator will then calculate the correctness of this comparison, which indicates the overall performance of the IR system. The most common metrics used are precision (1.1), the fraction of items retrieved that are relevant; recall (1.2), the fraction of items retrieved out of all relevant-labelled items; and F-measure (1.3), the harmonic mean of precision and recall [58].

$$P = \frac{|relevant \cap retrieved|}{|retrieved|} \tag{1.1}$$

$$R = \frac{|relevant \cap retrieved|}{|relevant|} \tag{1.2}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1.3}$$

Automatic keyphrase extraction is one form of IR that has been researched in recent years. The items being retrieved are keyphrases and the larger collection is the document with which the keyphrases are associated with. Keyphrases are n-grams (usually one to three words) that represent the main content or key topics of a document. They are highly beneficial in a number of fields, including information retrieval, summarisation and question-answering. In 1999, a landmark research KEA [71] successfully applied the technique of supervised machine learning [47] to the task of keyphrase extraction. KEA was evaluated against a dataset of documents taken from FAO [11], each of which came with a set of human-assigned keyphrases (the gold standard). For each document, the top ranked keyphrase candidates from KEA (up to 20) were compared with the expected gold standard and the number of direct string match-ups was used to indicate the system's overall performance. Many subsequent keyphrase extraction systems were evaluated in a similar fashion, including KEA++ [42], KP-Miner [10], Maui [41], Turney 2003 [63], Hulth 2003 [24], CollabRank [66] and DIKEA [67].

In the context of a keyphrase extraction system, the numerator of both the formulas for precision (1.1) and recall (1.2) counts the number extracted keyphrases that also appear in the gold standard, commonly referred to as the true positive (TP) value in statistics. By only recognising direct string matches, the TP calculated will tend to be unfairly low, thus lowering the score under both metrics. Given a set of candidate keyphrases extracted by the system (**C**) and a set of gold standard keyphrases (**G**), the TP value is formally defined by Equation (1.4).

$$TP = \sum_{c \in \mathbf{C}} \max_{g \in \mathbf{G}} match(c, g) \tag{1.4}$$

The function $match$ is the operative component in calculating TP. Conventionally, keyphrases are stemmed [15], accent folded (unicode characters are converted to their ASCII equivalent) and case folded (converted to lower case) prior to string matching. This allows small variations in the candidate keyphrases to still be counted towards the TP value (i.e. variations in inflection or grammar).

$$match(c, g) = \begin{cases} 1, & \text{if } stem_fold(c) == stem_fold(g) \\ 0, & \text{otherwise} \end{cases} \tag{1.5}$$

Conducted in this manner, automatic evaluation oversimplifies the process by assuming relevancy is purely black and white. Retrieved items are classified as either completely relevant or irrelevant. In practice, relevancy comes in many shades, spanning a full continuum between relevant and irrelevant, which is an important subtlety that traditional automatic evaluation methods fail to capture. In the case of keyphrase extraction, this exact matching criteria results in a highly pessimistic evaluation of the system. In other words, the precision, recall and F-measure scores calculated only act as an absolute lower bound of a system's true performance. When testing KEA++ [42], the authors noted that the average agreement between humans was only 38% when using this evaluation technique. They then went on to use the gold standard with low human agreement to justify the performance scores achieved by their own system. In fact, the problem lies in the evaluation technique itself, and not with the consistency of human keyphrase labelling.

The same matching criteria was also used in task 5 of the 2010 Workshop on Semantic Evaluation (SemEval-2010 [31]), where a total of 19 automatic keyphrase extraction systems competed against each other. Conductors of this workshop observed that the performance scores were all superficially low and calculated the human agreement to a poor $33.6\%$. They then correctly concluded that the exact string matching method used for evaluation only offers a lower bound for true performance. Semantics (the actual meaning behind phrases) must be taken into account in order to provide a more accurate judgement of keyphrase acceptability.

When semantics need to be considered in a comparison, a common metric to use is semantic relatedness [19], which computes the *distance* between two entities based on the relationship of their semantic content. Some well-known implementations include LSA [9], ESA [16], NGD [4] and WLM [70]. The problem with semantic relatedness is that it covers a broad spectrum of relationships, including antonymy. Therefore, it does not make for an appropriate $match$ function, as it relaxes the matching criteria too far. Several methods have been devised for approximate string matching that avoid using semantic relatedness, such as R-precision [73] and its successor Modified R-precision [30]. However, most of these techniques either only perform simple lexical-based string manipulation, or only consider synonymy, the simplest of semantics. As a result, current approaches fail to provide a suitable matching function. Furthermore, this research area lacks a suitable dataset for evaluation. Existing test collections are either targeted towards semantic relatedness (e.g. WordSimilarity-353 [13]), or were created with only lexical-based string manipulation in mind, such as those used in [73] and [30].

In this thesis, we will address the issue of how keyphrase extraction systems can be evaluated to reveal their true efficacy. The aim is to develop a new semantically-oriented approximate string matching criteria. One that is comparable to that of human judgements, but without the cost and energy associated with manual evaluation. Our contributions are threefold.

1. We precisely define and quantify substitutability – a measure of the replaceability of one term for another. This measure can be used as a semantically-oriented alternative to the traditional matching function in Equation (1.5), to mimic the judgement quality of a human. Based on this definition, a generic system is designed that can compute substitutability by mining external knowledge sources.

2. We develop two concrete systems based on our generic design – one backed by WordNet [45] and the other backed by Wikipedia (`https://www.wikipedia.org/`).

3. We devise a method for evaluating approximate string matching systems by isolating the task in question. The process involves both the construction of a human-labelled dataset, and the creation an assortment of metrics that can evaluate the agreement of a system with the dataset (i.e. its ability to mimic human judgements).

The remainder of this thesis is structured as follows. Chapter 2 provides an analysis for existing solutions to approximate string matching, including both semantic and lexical approaches. The first section of Chapter 3 formally defines substitutability and proposes a generic design for a system which can compute substitutability using an external knowledge source. Later sections of Chapter 3 present our two concrete implementations of the generic design, WordSub and WikiSub. Chapter 4 describes the creation of a dataset which isolates the task of computing substitutability, and introduces a range of metrics for evaluating the performance of systems that perform this task. Chapter 5 evaluates WordSub and WikiSub against each other and against existing approaches. It then presents some ways that our substitutability systems can be further optimised, both in terms of performance/agreement and speed. Finally, Chapter 6 concludes the thesis, with a summary of our contributions and several suggested areas of research for future work.

# Chapter 2

# Related Work

In this chapter, we discuss the various ways in which semantics can be introduced into string comparisons, briefly explaining the advantages and disadvantages of each method. Then we showcase a range of existing systems that have either been designed with approximate string matching in mind, or can easily be adapted to suit this task.

## 2.1 Semantic Relatedness

One way to *soften* the string matching criteria is by applying the notion of semantic relatedness or similarity [19], which is a distance metric that measures the likeness between two terms. Unlike exact string matching, semantic relatedness regards more than just the lexical representation of a term (the literal string of characters), but instead, as the name suggests, focuses on meaning or semantic content. This distance measure can be calculated in a variety of ways, many of which involve statistical analysis.

A well researched approach, which has yielded positive results in the past, is to extrapolate a set of topics or concepts associated with a term. The term can then be represented by a vector of weighted concepts, and subsequently calculate its relatedness to other terms using a similarity measure such as cosine similarity [57]. Latent Semantic Analysis (LSA [9]) is a system which employs this method. Purely statistical, LSA exploits word co-occurrence information by training itself on a large unlabelled corpus of text. It then constructs a words-by-documents co-occurrence matrix from the corpus. Finally, it performs dimensionality reduction by factorising the matrix using Singular Value Decomposition [18]. The dimensions that remain are assumed to be latent concepts, allowing terms to be compared by measuring their distance or similarity within this unified concept-space. Explicit Semantic Analysis (ESA [16]), a successor of LSA, leverages Wikipedia to produce a similar concept space. In this case, concepts are defined to be Wikipedia articles. As the name implies, ESA uses explicit concepts that have been collected and organised by humans, in contrast to the latent concepts *learnt* by LSA. This leads to a notable improvement in the correlation between the computed relatedness score and human judgements. Salient Semantic Analysis (SSA [20]) further improves upon this approach, by exploiting the annotated links available between Wikipedia articles. Instead of examining the distribution of words inside Wikipedia articles, SSA directly profiles terms by observing the co-occurring inter-article links (salient concepts) within a given window size.

The downside of vector-based methods is that they require an initial pre-processing step that is time consuming. For example, ESA has to build an inverted index of the entirety of Wikipedia, mapping each possible term to a set of Wikipedia articles. A different approach to measuring semantic relatedness is by inspecting the ease with which one entity can transform into another. The similarity metric used to calculate this measure is called the Normalised Information Distance (NID [37]), which is formally defined in Equation (2.1). $K(x)$ is a function that expresses the Kolmogorov Complexity [33] of $x$. In the context of strings, Kolmogorov Complexity is the length of the shortest possible representation of the string in some predefined universal description language. $K(x|y)$ is then the shortest possible program that can reproduce $x$ given $y$ (i.e. the minimum computational resources required to transform $y$ into $x$). Kolmogorov Complexity is not actually computable, but its upper bound can be estimated by using real-world compressors and compression algorithms (e.g. Gzip [17], LZSS [59], LZW [68], PPM [5] and DEFLATE [7]). The better the compressor, the closer the approximation is to the true value. Equation (2.2) shows the real-world approximation of NID, known as the Normalised Compression Distance (NCD [3]), for a given compressor $C$. A direct implementation of NCD is the Normalised Google Distance (NGD [4]) measure, which views Google (or any other search engine) as the compressor. Equation (2.3) demonstrates how this measure is calculated, where $f(x)$ signifies the number of web pages containing $x$ (returned by the search engine), and $f(x, y)$ is the number of web pages containing both $x$ and $y$. $N$ is simply a normalisation constant, normally set to the total number of returnable pages. For Google, a value above $10^{10}$ is suggested.

$$NID(x,y) = \frac{\max(K(x|y), K(y|x))}{\max(K(x), K(y))} \tag{2.1}$$

$$NCD(x,y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} \tag{2.2}$$

$$NGD(x,y) = \frac{\max(\log f(x), \log f(y)) - \log f(x,y)}{\log N - \min(\log f(x), \log f(y))} \tag{2.3}$$

A third approach for computing semantic relatedness is to mine the underlying graph/inter-linking structure formed by various external knowledge bases. WordNet::Similarity [53] is a free software package that contains a variety of different relatedness measures which utilise the relational links present in WordNet [45]. These include **wup** [72], **lch** [36] and **hso** [22]. WordNet maintains a hierarchy of concepts/synsets (groups of tightly-coupling synonyms), which follow an is-a relationship. **wup** calculates the relatedness of two synsets, $X$ and $Y$, by finding the depth of their least common subsumer (LCS) – the most specific synset which is still a generalisation of both $X$ and $Y$. **lch** performs the same task by finding the shortest path between the two synsets. **hso** is similar to **lch**, but also tries to minimise the number of directional changes along the path (e.g.

moving up and then down in the hierarchy). Other implementations of this approach include WikiRelate! [60], which exploits the hierarchically-organised structure of Wikipedia categories; and Wikipedia Link Measure (WLM [70]), which measures relatedness by inspecting hyperlinks within Wikipedia articles. WLM stands out from other implementations in that it does not rely on finding a path. Instead, it compares two articles by directly examining their shared hyperlinks (both incoming and outgoing). The greater the overlap, the higher the relatedness is. This allows semantic relatedness to be calculated using Equation (2.4), where $X$ and $Y$ are the sets of articles that link to $x$ and $y$ respectively, and $W$ represents all the articles in Wikipedia.

$$WLM(x,y) = \frac{\log(\max(|X|,|Y|)) - \log|X \cap Y|}{\log|W| - \log(\min(|X|,|Y|))} \tag{2.4}$$

All three approaches successfully capture semantic information when comparing terms against each other. But an approximate string matching criteria based on semantic relatedness compensates too far in the other direction (i.e. it is not strict enough). Semantic relatedness covers too broad a range of relationships (including antonymy). When we evaluate keyphrase extraction systems, the $match$ function should return a non-zero value only for inputs that are either lexically equivalent or the candidate is a suitable replacement for the gold standard. The higher the value, the better the replacement is, up to a perfect match of $1.0$. Thus, Section 3.1.1 will introduce the notion of substitutability as a subclass of semantic relatedness – covering a stricter subset of relationships. It is also important to note that semantic relatedness is a symmetric measure (i.e. A is related to B the same amount as B is related to A), which may not be desirable. Section 3.1.1 formally defines substitutability and discusses the potential benefits of an asymmetric measure. Sections 3.2 and 3.3 introduce two systems which implements a substitutability-oriented approximate matching criteria using a path-based method, similar to the one described in this section, but with a stricter matching criteria and new ways for quantification.

## 2.2 Existing approaches

Problems with exact string matching were realised over a decade ago. In 2000, Barker and Cornacchia [1] disclosed that the task of evaluating the quality of a set of keyphrases is a highly subjective task. At its core, the quality of keyphrases should reflect their ability to give the reader a rough understanding of the contents of a document. The subtleties involved in this judgement make it very difficult to compute automatically. Instead, Barker and Cornacchia proposed an alternative evaluation method which involves human judgements. But in the end, they concluded this type of evaluation should be avoided as it is a difficult, time and energy-consuming processing.

The difficulty of the task is further compounded by the fact that authors have a tendency of assigning gold standard keyphrases that do not occur in the actual document. This essentially puts a cap on the highest possible overlap between a set of automatically extracted keyphrases and the gold standard – most keyphrase extraction systems can only reproduce keyphrases from existing phrases in the document. Barrière and Jarmasz (2004 [2]) determined that automatic evaluators must go beyond exact string matching in order to yield useful and conclusive results. They introduced a different matching criteria based on Pointwise Mutual Information (PMI [62]). This marks the first occurrence of a semantically-oriented approach to evaluating keyphrase quality.

PMI (2.5) is a statistical measure that indicates the co-occurrence between two terms, which has been shown to be a good estimator of semantic relatedness [61]. A value of zero indicates two statistically independent terms, negative values indicate a lack of co-occurrence, while a positive value indicates high co-occurrence. To adapt this measure for the $match$ function from Chapter 1, it must be fed through a normalising function such as a logistic sigmoid as illustrated in Equation (2.6). There are two downsides to this technique. First, like other statistical approaches, it requires a long set up time to process a vast unlabelled corpus of text (one terabyte in the case of [62]). Second, co-occurrence captures all aspects of semantic relatedness, including undesirable relationships such as antonymy [12].

$$PMI(x, y) = \log \frac{p(x \cap y)}{p(x) \cdot p(y)} \tag{2.5}$$

$$match(x, y) = sigmoid(PMI(x, y)) = \frac{1}{1 + e^{-PMI(x,y)}} \tag{2.6}$$

In 2009, Zesch and Gurevych [73] also reiterate the shortcomings of exact string matching, stating that it is known to underestimate performance as perceived by human judges. So as part of their newly proposed evaluation metric, **R-precision**, the authors introduced three approximate matching strategies. The first strategy, MORPH, simply accounts for morphological variants (grammatical inflections) between two terms, which is something that conventional evaluation methods already achieve via stemming. The other two strategies are INCLUDES and PARTOF, which account for partially overlapping phrases at the level of words. INCLUDES addresses situations where the extracted candidate *includes* the gold standard, while PARTOF is for when the candidate is *part of* the gold standard.

Being a purely lexical-based method, the approximate matching criteria used in R-precision do not rely on an external knowledge source and are able to return a result very quickly by performing basic string manipulation and comparisons. However, by disregarding semantics, this type of matching can over-value semantically dissimilar terms such as "red wine" vs. "white wine". Despite this, the authors showed that accounting for partial phrasal overlaps provides a definite improvement to exact string matching. They evaluated each approximate matching strategy by having four human judges review term pairs that passed the matching criteria, either accepting or rejecting the match. While this does evaluate whether a *match* is acceptable, it neglects to check whether a *mismatch* is acceptable. Because of this, MORPH trivially performs the best by returning the least number of matches. In Section 4.2, we propose a range of evaluation metrics to test the acceptability of both matches and mismatches against human judgements.

$$R - p(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x} \cap \mathbf{y}|}{\max(|\mathbf{x}|, |\mathbf{y}|)} \tag{2.7}$$

The matching strategies used by R-precision, whilst less strict than exact string matching, are still simple binary acceptance criteria. In 2010, R-precision was generalised by Kim, Baldwin and Kan [30] to be the fraction of overlapping words between two terms over the length (number of words) of the longer term (as shown in Equation (2.7)). This makes it a viable formula for the $match$ function. R-precision, in this new form, was later used by Joty et al. (2012 [26]) for evaluating systems that perform automatic topic segmentation and labelling, proving such approximate matching techniques can be applied across a wide range of evaluation tasks in the field of IR, especially those that concern natural language processing.

$$mod.\ R - p(\mathbf{x}, \mathbf{y}) = \frac{\sum_{w_i \in \mathbf{y}} \begin{cases} \frac{1}{|\mathbf{y}| - i}, & \text{if } w_i \in \mathbf{x} \\ 0, & \text{otherwise} \end{cases}}{\sum_{n=1}^{|\mathbf{y}|} \frac{1}{n}} \tag{2.8}$$

Kim et al. also introduce an enhanced version of R-precision which they coined **Modified R-precision**. Unlike R-precision, which is indifferent to word order, Modified R-precision weights each overlapping word from both phrases based on its position. Equation (2.8) formally defines this measure, where $\mathbf{y}$ is assumed to be the longer of the two phrases (if they are different in length) and $w_i$ is the $i^{th}$ word in $\mathbf{y}$ (indexed from zero). This measure is motivated by the assumption that the head noun is usually at the end of a noun phrase. Therefore, the further a word is from the head noun, the lower its importance is towards the matching, and thus it receives a decreased weighting. For example, comparing ″applied science″ to ″natural science″ would yield a match of $\frac{2}{3}$ under Modified R-Precision, where as the standard R-precision would yield a match of $\frac{1}{2}$.

Kim et al. also adapted a range of n-gram based metrics, from the field of machine translation (MT) and summarisation, to perform approximate matching – **BLEU** [51], **NIST** [40], **ROUGE** [38] and **METEOR** [34].

BLEU (Bilingual Evaluation Understudy) was traditionally used as an evaluation metric for measuring the similarity between a candidate translation against a reference (gold standard) translation. It does this by counting the number of overlapping n-grams, with an additional penalty if the candidate is shorter than the reference. Equations (2.9) to (2.11) demonstrates how the measure is calculated when adapted to the task of approximate string matching. Equation (2.9) calculates the n-gram precision for a particular n-gram length of $n$, which is simply the fraction of overlapping n-grams over the total. Equation (2.10) is the brevity penalty used when phrases are of different length. Finally, Equation (2.11) produces the actual measure, which takes into account all possible n-gram lengths. Again, $\mathbf{y}$ is assumed to be the longer of the two phrases, in terms of the number of words.

$$p_n(\mathbf{x},\mathbf{y}) = \frac{\sum_{ngram \in \mathbf{x}} \begin{cases} 1, & \text{if } ngram \in \mathbf{y} \\ 0, & \text{otherwise} \end{cases}}{|\mathbf{x}| - n + 1} \tag{2.9}$$

$$BP(\mathbf{x},\mathbf{y}) = \begin{cases} \mathrm{e}^{1-|\mathbf{y}|/|\mathbf{x}|}, & \text{if } |\mathbf{x}| \leq |\mathbf{y}| \\ 1, & \text{otherwise} \end{cases} \tag{2.10}$$

$$match_{\text{BLEU}}(\mathbf{x},\mathbf{y}) = BP(\mathbf{x},\mathbf{y}) \cdot \exp\left(\sum_{n=1}^{|\mathbf{x}|} \frac{1}{|\mathbf{x}|} \log p_n(\mathbf{x},\mathbf{y})\right) \tag{2.11}$$

NIST (National Institute of Standards and Technology) is calculated very similarly to BLEU, except $p_n$ (n-gram precision) is altered to weigh each n-gram based on their occurrence. Higher weightings are given to n-grams which occur less frequently, as they are reasoned to be more distinct and thus provide greater informational value. Conversely, frequently occurring n-grams have lower weightings.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a package which includes four metrics for determining the quality of a document summary (the candidate) when compared to an ideal summary created by humans (the gold standard). ROUGE-N is one of these metrics, which measures the n-gram recall between summaries for n-grams of a specific length N. When adapted for the task of approximate string matching, ROUGE closely resembles the n-gram precision formula ($p_n$) from BLEU as shown in Equation (2.12). The difference is that ROUGE iterates n-grams over the longer phrase $\mathbf{y}$, whilst $p_n$ iterates over the shorter phrase $\mathbf{x}$. In [30], Kim et al. experiments exclusively with ROUGE-1 (i.e. only considers single word overlap). A variation of this approximate matching strategy was also used to evaluate the TextRank system [44].

$$match_{\text{ROUGE}}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{ngram \in \mathbf{y}} \begin{cases} 1, & \text{if } ngram \in \mathbf{x} \\ 0, & \text{otherwise} \end{cases}}{|\mathbf{y}| - n + 1} \tag{2.12}$$

METEOR (Metric for Evaluation of Translation with Explicit Ordering) evaluates a candidate translation by finding explicit unigram (single word) matches between the candidate and reference (gold standard) translation. The initial step is to find a *word alignment* between the candidate and the reference strings. This alignment maps one word from the candidate to exactly one word from the reference. In graph theory, this is known as a maximum cardinality bipartite matching [69], where the nodes are words and edges are potential mappings. Word mappings are assigned over three passes, with the required criteria for a potential mapping being increasingly relaxed for each pass. In the first pass, a word can only be mapped to other words that have the exact same string of characters. In the second pass, a word is allowed to be mapped to other words that have the same stem, as determined by the Porter stemmer [54]. For the final pass, semantics are taken into consideration by allowing word mappings that represent synonyms. This is achieved by examining synsets in WordNet, which are sets of synonyms that conforms to a distinct concept. In Section 3.2, we introduce a system whose opening stage closely resembles the alignment step in METEOR, but our system creates mappings at the phrase/n-gram level whenever possible.

After finding the maximum word alignment, METEOR calculates the precision and recall between the two strings using Equations (2.13) and (2.14) respectively, where $maxalign$ returns the number of word-to-word mappings in the maximum valid word alignment. These are then combined using a parameterised harmonic mean (or weighted f-measure [64]) as per Equation (2.15). As stated previously, $\mathbf{y}$ is assumed to contain the same number or more words than $\mathbf{x}$.

$$P(\mathbf{x},\mathbf{y}) = \frac{maxalign(\mathbf{x},\mathbf{y})}{|\mathbf{x}|} \tag{2.13}$$

$$R(\mathbf{x},\mathbf{y}) = \frac{maxalign(\mathbf{x},\mathbf{y})}{|\mathbf{y}|} \tag{2.14}$$

$$F_\alpha(\mathbf{x},\mathbf{y}) = \frac{P(\mathbf{x},\mathbf{y}) \cdot R(\mathbf{x},\mathbf{y})}{\alpha \cdot P(\mathbf{x},\mathbf{y}) + (1-\alpha) \cdot R(\mathbf{x},\mathbf{y})} \tag{2.15}$$

METEOR takes word ordering into consideration by applying a *fragmentation* penalty to the final score as formulated by Equation (2.16). Fragmentation is defined to be the fraction of the least number of *chunks* possible in a maximum word alignment ($maxalign_{ch}$) over the total number of mappings ($maxalign$). A chunk is specified as an adjacent sequence of words that occur in both strings and have been mapped together in a maximum word alignment. For example, consider a maximum word alignment between `ABXY` and `ABCXY`, where each character represents a word. The maximal mapping will produce two chunks, `AB` and `XY`, out of four total mappings. Therefore, the fragmentation value will be $0.5$. A higher fragmentation value results in a larger penalty, as defined by Equation (2.17), which is further parameterised by $\beta$ and $\gamma$. Equation (2.18) illustrates how the fragmentation penalty and weighted harmonic mean is combined to form the final METEOR measure of similarity. Lavie and Agarwal (2007 [35]) later tuned METEOR's three system parameters for various tasks. The optimal values for measuring the similarity between English phrases are as follows: $\alpha = 0.81$, $\beta = 0.83$ and $\gamma = 0.28$.

$$frag(\mathbf{x}, \mathbf{y}) = \frac{maxalign_{ch}(\mathbf{x}, \mathbf{y})}{maxalign(\mathbf{x}, \mathbf{y})} \tag{2.16}$$

$$pen(\mathbf{x}, \mathbf{y}) = \gamma \cdot frag(\mathbf{x}, \mathbf{y})^{\beta} \tag{2.17}$$

$$match_{\text{METEOR}}(\mathbf{x}, \mathbf{y}) = (1 - pen(\mathbf{x}, \mathbf{y})) \cdot F_{\alpha}(\mathbf{x}, \mathbf{y}) \tag{2.18}$$

In Section 5.2.1, we will implement a selection of the measures covered in this section, and compare them against the two systems we introduce in Chapter 3. This will be done by evaluating each system's ability to correctly compute the replaceability (or substitutability) of a candidate term for a gold standard term, based on each measure's agreement with human judgements.

# Chapter 3

# Design and Implementation

In this chapter, we present the details of designing a semantically-oriented approximate string matching system based on the notion of substitutability – referred to as a substitutability system from henceforth. Given this generic design, we will then demonstrate how various external knowledge resources can be used to implement this design. Two specific knowledge sources will be showcased – WordNet, a database of English words and common phrases; and Wikipedia, a vast online collection of interlinking articles, which cover all aspects of human knowledge.

## 3.1 General design

### 3.1.1 Overview

Substitutability is a measure of how suitable a word (or phrase) is when used in place of another. The phrase to be replaced is known as the substitutee, and the phrase to replace it with is known as the substitute. It is a subclass of semantic relatedness that describes a stricter subset of relationships. A pair of substitutable phrases implies that they are also semantically related, but a pair of semantically related phrases does not necessarily imply substitutability. For our purposes, we will consider substitutability to be a score between 0.0 and 1.0, given to a specific substitute-substitutee pair. 0.0 would imply no substitutability (i.e. the substitute is not suitable to be used in place of the substitutee at all) and 1.0 would imply perfect substitutability (i.e. the substitute is perfectly suitable to be used in place of the substitutee).

A trivial design for a substitutability system is one which gives a perfect 1.0 score only to substitute-substitutee pairs that are identical (i.e. two equivalent strings), while simply assigning 0.0 to all other pairs. Such a system is extremely conservative, in the sense that every time it produces a 1.0 score, it is guaranteed to be correct, but many of the pairs that are scored 0.0 will be undervalued.

A simple example is a plurality pair, such as using `chair` in place of `chairs`. The pair clearly have perfect (or near-perfect) substitutability; they are simply inflections (the modification of a word for grammatical reasons, such as tense, number and person) of one another. But the trivial system defined previously would score it incorrectly at 0.0.

A quick fix to this issue is to first reduce both terms to their stems (removing any inflections) prior to comparing them. Note that a stem need not be the same as a word's morphological root. It is common for a stem to not be a valid root, and a broader range of related words tend to be mapped to the same stem. For example, the stem of `management` is `manag`, while its morphological root is `manage`. Morphological roots also remove any prefixes, which is especially problematic with negatively modified words such as `unfinished`, which has the root `finish`, thus reversing its meaning. A stem on the other hand would only reduce the word to `unfinish`, making it perfect for this task. In fact, this is the most common method for matching terms used to evaluate keyphrase extraction systems (and still the most standard approach to date, see Chapter 2).

A less trivial pair to consider is using `run` in place of `sprint`. Lexically speaking, the two words are completely different and reducing them down to their stems will not help. However, as humans, we understand that they have similar meanings. Sprinting is a form of running. Therefore, the pair `run-sprint` is clearly substitutable, or at least partially substitutable (i.e. a score somewhere between, but exclusive of 0.0 and 1.0). However, it is unclear what substitutability score to give exactly.

What we need is a strict, quantifiable definition of substitutability that is robust enough to be applicable in any situation. Below is a concise description of what we believe substitutability is:

**"Substitutability is a context-free measure of how much information is retained when one phrase is used in place of another, whilst making as few assumptions as possible. "**

The remainder of this subsection expands on this definition into a set of rules and restrictions for exactly how we aim to measure substitutability.

#### Context and disambiguation

The substitutability of a substitute is only measured in relation to the substitutee alone, disregarding any surrounding text that the terms may appear in. If either of the terms have ambiguous meanings, then it should be disambiguated against the other term such that it maximises substitutability. This means that the context in which the substitutee appears in has no influence on the substitutability of the substitute in question. While this may not always be correct, it does help to restrict and simplify the problem so that any particular substitute-substitutee pair will always have the same substitutability score.

#### Grammar

We are only concerned with how well a term can substitute another in regards to their semantic meaning. Whether the replacement results in valid grammar or syntax is irrelevant. This is again because we are only measuring the terms (substitute/substitutee) on their own and ignoring the larger context that they may be part of.

### Information loss/retention

When substituting one term in place of another, it is inevitable that some of the meaning (or information) will be lost. Exactly how much information is lost (or how much information is retained) is a good indicator of how suitable the substitution is. Therefore, the substitutability of a substitute-substitutee pair should strongly reflect the information retention of the substitution.

By this definition, the plurality pair `chair-chairs` would have perfect information retention and thus a perfect substitutability score of 1.0 (recall that grammar is irrelevant). In contrast, the pair `run-sprint` is not a perfect substitution. Using `run` in place of `sprint` no longer conveys the speed of the run (i.e. this information is lost in the substitution). A more extreme example of information loss is the pair `emotional-excited`, which arguably loses much more meaning than `run-sprint` – there are countless other emotions besides excitement. From this, we should expect to see the pair `run-sprint` to receive a higher substitutability score than `emotional-excited`, but definitely less than 1.0 as neither are perfect. In Section 3.1.3, we will show exactly how information loss can be quantified.

### Information gain

Just as a substitute can lose information about a substitutee, it can also add extra information. When this happens, we say that the substitute has made an assumption on the substitutee. Consider the pair `cake-food`, that is, we want to use `cake` in place of `food`. While no information is actually lost in the substitution, the meaning has completely changed. The substitute has added extra information by assuming the type of food to be cake. Therefore, substitutes that make assumptions should be avoided when possible, as they can completely change the meaning if used in place of the substitutee. In other words, information gain should be heavily penalised.

A noteworthy feature of substitutability, as a result of this rule, is that it is an asymmetric relationship, unlike most other semantic relatedness measures (see Chapter 2). For example, the pair `food-cake` is more suitable than the pair `cake-food`. That is, `food` can be used in place of `cake`, because it does not make any assumptions (add extra information) towards `cake`.

### 3.1.2 Targets and sub-targets

For our design of a substitutability system, we will assume that all inputs come in the form of two phrases in plain text. The first one being the substitute and the second one being the substitutee. In order to utilise and mine external knowledge databases (referred to as knowledge bases henceforth), we must first match the plain text input to a specific entity in the knowledge base. An entity represents a basic unit of knowledge, which could be an article, entry, page, node etc., specific to a particular resource. We have decided to call such matchings **targets**, the reason for which will become obvious later on in this section. Sections 3.2 and 3.3 will demonstrate how this step is performed for WordNet and Wikipedia.

Unfortunately, most English terms have multiple meanings (they are ambiguous), and we must consider all of these to ensure we maximise substitutability. Also, because grammar is irrelevant, we will stem each term prior any matching. So each term will in fact be matched to a set of targets (called a **target set**), the details of which are once again specific to each individual knowledge base.

But what if a term fails to match any entity in a knowledge base? As is often the case with English phrases with more than one word. When this occurs, we should break down the phrase into its components (or aspects [6]) in an attempt to get a successful match. If the term in question is a single word, then nothing can be done at this point and the entire process ends here, returning a default substitutability score of 0.0 to be on the safe/conservative side. Section 5.3.6 will discuss a potential way of getting around this problem by using multiple knowledge bases together.

To break down a phrase, we first remove any stop words (functional words that serve only syntactic purposes in a phrase) before splitting up the phrase into non-overlapping aspects. The list of stop words will be taken from the same list used in DIKEA [67]. An attempt will then be made to match these aspects to entities in the knowledge base. If successful, the matching entities become **sub-targets**. If a match still cannot be found, each aspect is broken down further (or the phrase is split at different points) until either all aspects are matched up, or the phrase has been reduced to single words. At which point the system once again returns a conservative score of 0.0. Algorithms 1 and 2 show exactly how a phrase is matched to its targets and/or sub-targets. The sub-routine MATCHTARGETS attempts to match a phrase (list of words) to a specific set of targets in the knowledge base, returning an empty set if no match is found.

Consider a simple example where only one of the phrases cannot be matched. We want to test the substitutability of using `stationery` in place of `pen and paper`. For this example, we will assume that `pen and paper` cannot be matched to an entity in the knowledge base. So it must be broken down by first removing the stop word "and", and then splitting the phrase in half to produce the following aspects:

```
[ Pen ] [ Paper ]
```

Suppose both these aspects are able to be matched. We now have a single target set for the substitute and two sub-target sets for the substitutee. At this point, every unique pair of target sets (one from the substitute side and one from the substitutee) is tested against each other to produce a substitutability score for each pair. We then select a subset of these pairs that produced the highest scores, such that each target set is covered at least once. This ensures we maximise substitutability without leaving out any part of the input.

**Algorithm 1** Finds the targets and sub-targets that can successfully be matched to a phrase whilst minimising the number of aspects. Returns a list of target sets or an empty list if no matching is found.

```
procedure FindTargets($phrase$)
    $ts \leftarrow$ MatchTargets($phrase$)
    if $ts$ is empty then
        $p \leftarrow$ RemoveStopWords($phrase$)
        for $i \leftarrow 1, p.length$ do
            $tss \leftarrow$ FindAspects($p, i$)
            if $tss$ is not empty then
                return $tss$                    ▷ aspects matched to sub-targets
            end if
        end for
        return [ ]                              ▷ no possible match for $phrase$
    else
        return [$ts$]                           ▷ $phrase$ directly matched to targets
    end if
end procedure
```

**Algorithm 2** Finds the aspects in a phrase that can be matched to target sets and returns a list of those target sets. An empty list is returned if no match is found.

```
procedure FindAspects($p, n$)                              ▷ splits $p$ into $n$ aspects
    if $n == 1$ then
        $ts \leftarrow$ MatchTargets($p$)
        if $ts$ is empty then
            return [ ]                              ▷ $p$ could not be matched to targets
        else
            return $[ts]$
        end if
    else
        for $i \leftarrow 1, p.length - n + 1$ do
            $psub1 \leftarrow p[0, i]$        ▷ front end of $p$ up to, but excl. $i$th word
            $ts \leftarrow$ MatchTargets($psub1$)
            if $ts$ is empty then
                continue
            end if
            $psub2 \leftarrow p[i, p.length]$        ▷ back end of $p$ from the $i$th word
            $tsrest \leftarrow$ FindAspects($psub2, n - 1$)
            if $tsrest$ is not empty then
                return $[ts] + tsrest$                        ▷ all targets matched
            end if
        end for
        return [ ]                                  ▷ $p$ could not be matched to targets
    end if
end procedure
```

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{3.1}$$

The final score is calculated as the harmonic mean [29] of the selected subset of scores. Harmonic mean is calculated using Equation (3.1), where each $x_i$ is a score out of $n$ scores. In the current example, there is only one possible selection of pairs, which are as follows:

$$\texttt{stationery} \rightarrow \texttt{pen} = 0.5$$
$$\texttt{stationery} \rightarrow \texttt{paper} = 0.3$$

The substitutability scores are for demonstration purposes only. The final score for this example will be $0.375$.

To show how this process works for more complicated scenarios, consider an input where the substitute has to be split into three aspects and the substitutee has to be split into two.

$$\texttt{[ A ] [ B ] [ C ]} \rightarrow \texttt{[ X ] [ Y ]}$$

After calculating the substitutability score of each unique pair of target sets, we can order them by their scores as follows:

$$\texttt{[ B ]} \rightarrow \texttt{[ X ]} = 1.00$$
$$\texttt{[ C ]} \rightarrow \texttt{[ X ]} = 0.85$$
$$\texttt{[ C ]} \rightarrow \texttt{[ Y ]} = 0.80$$
$$\texttt{[ A ]} \rightarrow \texttt{[ Y ]} = 0.60$$
$$\texttt{[ B ]} \rightarrow \texttt{[ Y ]} = 0.35$$
$$\texttt{[ A ]} \rightarrow \texttt{[ X ]} = 0.10$$

We then pick a subset from this list, going from top to bottom, with the highest substitutability that covers each target set at least once. In this case, the top four substitutions will be selected, giving a final result of $0.79$ (2dp).

A useful property of the harmonic mean is that if any of the scores are $0$, then the final score will also be $0$. One cannot use the harmonic mean formula when zeros are involved, as this would lead to division by zero. Instead, harmonic mean is simply defined to be zero when the list of rates being averaged also contains at least one zero. This should ensure that we will not accidentally overvalue a substitute, just because it partially overlaps the substitutee at the word level, as shown in the following example.

$$\texttt{[ good ] [ dog ]} \rightarrow \texttt{[ bad ] [ dog ]}$$

Many lexical-based approximate string matching algorithms will give this example a non-zero score, even though the two phrases are clearly not substitutable; they just happen to both share the word ”dog”. Using our method, we expect to produce a list of scores as follows:

$$\begin{aligned}
\texttt{[ dog ]} \rightarrow \texttt{[ dog ]} &= 1.0 \\
\texttt{[ good ]} \rightarrow \texttt{[ dog ]} &= 0.0 \\
\texttt{[ dog ]} \rightarrow \texttt{[ bad ]} &= 0.0 \\
\texttt{[ good ]} \rightarrow \texttt{[ bad ]} &= 0.0
\end{aligned}$$

As you can see, there is no way to pick a subset from this list that does not contain a non-zero score and still cover each target set. Therefore, it would receive a final substitutability score of $0.0$ as expected.

Algorithm 3 shows exactly how targets are used to calculate the overall substitutability of two phrases. Most of the work is done in the sub-routine SEARCH, which is covered in Section 3.1.4.

**Algorithm 3** Calculates the substitutability of using $phrase1$ in place of $phrase2$. Returns a real value between $0.0$ and $1.0$.

```
procedure SUBSTITUTABILITY($phrase1, phrase2$)
    $p1 \leftarrow$ TOWORDS($phrase1$)            ▷ split phrase into list of words
    $p2 \leftarrow$ TOWORDS($phrase2$)
    $tss1 \leftarrow$ FINDTARGETS($p1$)
    $tss2 \leftarrow$ FINDTARGETS($p2$)
    if $tss1$ or $tss2$ is empty then
        return $0.0$            ▷ no targets found for one or both phrases
    else
        $subs \leftarrow [\ ]$
        for all $ts1$ in $tss1$ do            ▷ test all possible pairs
            for all $ts2$ in $tss2$ do
                $score \leftarrow$ SEARCH($ts1, ts2$)
                $sub \leftarrow \{score : score, from : ts1, to : ts2\}$
                append $sub$ to $subs$
            end for
        end for
        $allts \leftarrow \{tss1 + tss2\}$            ▷ set of all targets
        $topscores \leftarrow [\ ]$
        sort $subs$ by descending order of $sub.score$
        for all $sub$ in $subs$ do
            if $allts$ contains $sub.ts1$ or $sub.ts2$ then
                remove $sub.ts1$ and $sub.ts2$ from $allts$
                append $sub.score$ to $topscores$
                if $allts$ is empty then            ▷ all target sets covered
                    break
                end if
            end if
        end for
        return HARMONICMEAN($topscores$)
    end if
end procedure
```

### 3.1.3 Transitions and information loss/gain

The previous section shows how the input terms can be matched to entities in a knowledge base to become targets. The next step is to find the link or relationship between targets. For this to work, we require the knowledge base to have relational links between its entities. These may come in the form of incoming and outgoing links on a page, or a hierarchical structure for categorising entities. Either way, we need a method of **transitioning** from one entity/target to another. Sections 3.2 and 3.3 will provide exact implementation details for using WordNet and Wikipedia.

**Transitions** between entities convey a direct substitution, which means that information is either lost, gained or in some cases, unchanged. Based on this information change, we assign a score to the transition (the **transition score**) that is essentially the substitutability of the two entities that are linked by the transition in question. In order to measure information change, we also require the knowledge base to have some statistical summary or metadata on its relational links, which we can use as an indicator towards the type and quality of the transition.

In the design of a substitutability system, we have categorised all transitions into four types – **Same**, **Similar**, **Specialisation** and **Generalisation** – which will be explained in depth by the remainder of this section. We will also recommend appropriate transition scores for each type of transitions. These scores act as tunable parameters in a substitutability system and is the key to quantifying substitutability.

#### Same

The first type of transition describe entities which have perfect (or near-perfect) substitutability. Graphically, we have chosen to represent Same transitions flat along the x-axis to convey no information change.

One form of this transition occurs between entities that are grammatical inflections of each other. Since grammar is irrelevant, such transitions

Figure 3.1: Perfect substitutions.

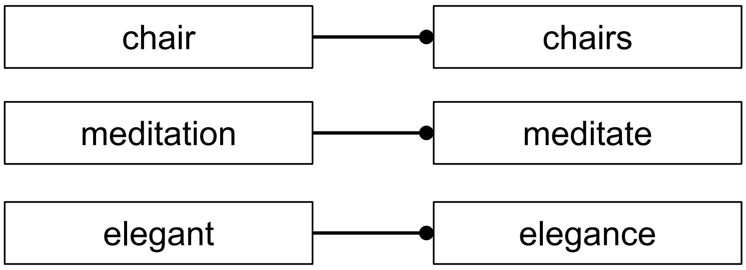
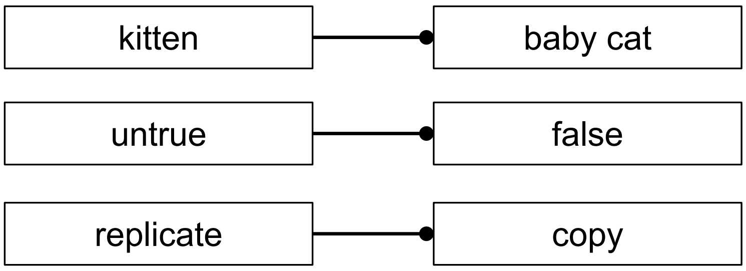


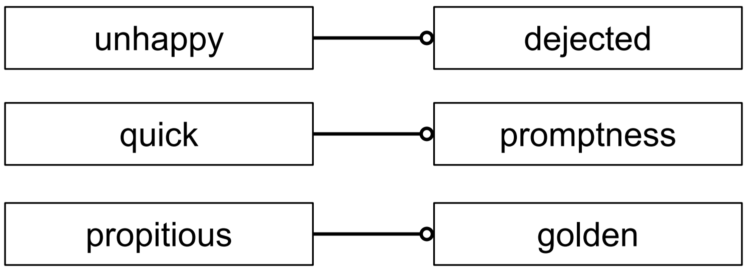
Figure 3.2: Near-perfect substitutions.



are considered perfect with a transition score of $1.0$ (see Figure 3.1). But there is one constraint. When using only this form of the Same transition, it is generally safe to assume a perfect transition score of $1.0$. But when combined with other transitions, the meaning will likely drift away from the original term. To avoid this, we only allow perfect Same transitions that directly link from/to entities that exist within target sets. These transitions are still allowed between non-target entities, but will be treated as **Similar** (discussed in the next transition type) transitions instead.

Another form of the Same transition occurs between entities which have the same meaning in each other's context, but are lexically different (e.g. synonyms) – see Figure 3.2. While these terms may have near-perfect substitutability, the meaning of the original entity may again drift over numerous transitions. So to discourage this behaviour, we will include a slight penalty to this form of the Same transition – giving it a transition score of $0.99$.

Figure 3.3: Similar transitions/substitutions.



### Similar

The second type of transition describe entities which have good substitutability, but are not perfect. Several transitions of this type could significantly drift the meaning of the original entity, so we will assign a larger penalty. A transition score between $0.5$ and $0.8$ should be appropriate, but the specifics will depend on the particular knowledge base used.

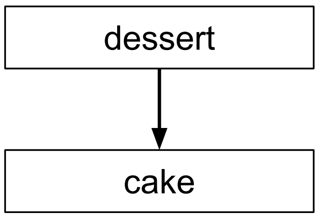
This type of transition can be difficult to identify in many knowledge bases and therefore may be omitted in concrete implementations. For example, WordNet, due to its dictionary-like nature, has plenty of relational links between entities that conform to the Similar transition. Wikipedia on the other hand, does not provide any obvious ways to do the same. We explored the idea of following hyperlinks in the "See also" sections of Wikipedia articles, but many of these were unrelated to the original article in terms of substitutability.

Figure 3.3 are some examples of the Similar transition.

### Specialisation

The third type of transition describe entities that are specialisations of another. More specifically, the entity acting as the substitutee is a specialisation of the entity acting as the substitute. Specialisation implies an is-a or is-a-type-of relationship, and unlike the previous two transitions, is asymmetric. This type of transition loses information, which may seem

Figure 3.4: A simple Specialisation transition/substitution.



counter-intuitive at first. But recall that all transitions move from the substitute side (e.g. `dessert`) to the substitutee side (e.g. `cake`). Therefore, the substitute (being the more general term) would lose some of the meaning of the substitutee if used in place of it.

Graphically, we have chosen to represent Specialisation transitions as moving downwards, to convey the information loss. See Figure 3.4.

Exactly how much information is lost will depend on how *common* the specialisation is. The easiest way to determine commonality is by observing the *breadth* of the specialisation. Given two terms, `A` and `B`, if `B` is very commonly a type or form of `A`, then little information is lost since the reader could potentially infer what `B` was when `A` is used in its place. In contrast, if `B` is just one of many (say 30 or more) types or forms of `A`, then a lot more information is lost about `B` when `A` is used in its place.

Figures 3.5 and 3.6 are some real life examples. In the first scenario, `lynx` (a type of wildcat) does not have many specialisations, so a transition from `lynx` to `bobcat` or `caracal` would have relatively low information loss. In the second scenario, `transport` has a large number of specialisations, so a transition to any of those entities (e.g. `transport-truck`) would instigate a greater loss of information than the first scenario. Thus, it should receive a lower transition score.

The breadth of a Specialisation transition from `A` to `B` is then the total number of Specialisation transitions that originate from `A`. If the breadth is small, we want to give the transition a high transition score. In fact, if the breadth is one, then the transition could be considered a **Same** transition.

Figure 3.5: Specialisation transition/substitution with narrow breadth.

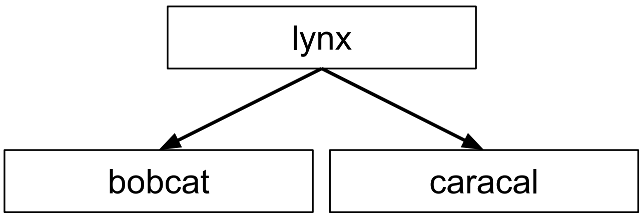
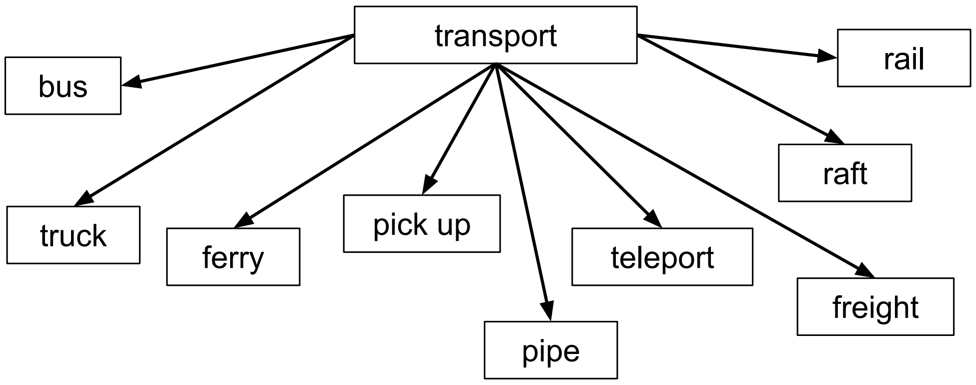


Figure 3.6: Specialisation transition/substitution with wide breadth.



In these special cases, we will give it a transition score of $0.99$, which is the highest score we will allow for a Specialisation transition: $Sp_{max} = 0.99$. As for the other end of the spectrum, we need to decide on the greatest penalty (or lowest score) that would be assigned to a transition of effectively infinite breadth. After some trial-and-error testing, $0.2$ was determined to be an appropriate minimum transition score: $Sp_{min} = 0.2$.

Now we can use Equation (3.2) to calculate transition scores based on the breadth, thus quantifying the information loss in Specialisation transitions. Note that the key coefficient is the inverse cube-root of the breadth, this is to dampen the penalty as the breadth increases.

$$score = Sp_{min} + \frac{1}{\sqrt[3]{breadth}} \cdot (Sp_{max} - Sp_{min}) \tag{3.2}$$

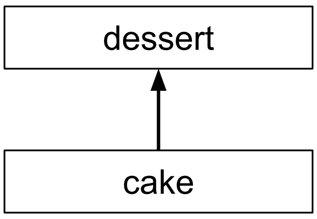
For the previous two examples (Figures 3.5 and 3.6), the transition score from `lynx` to `bobcat` will be $0.83$ (2dp), while a transition from `transport` to `truck` will score $0.58$ (2dp). In reality, the breadth of the latter transition would be much greater, so an even lower score is expected for `transport-truck`. For example, with a breadth of 50, the score would be reduced to $0.41$ (2dp).

### Generalisation

The fourth and final type of transition describes entities that are generalisation of others. This is simply a Specialisation transition in the reverse direction. Just as specialisations lose information, generalisations gain information. Graphically, we represent this as moving upwards, to convey an increase in information (see Figure 3.7).

According to our definition of substitutability, this should be heavily penalised, as it indicates that the entity acting as the substitute has made an assumption towards the substitutee. For example, a transition from `truck` to `transport` (opposite to the previous example) would assume the exact type of transport, adding extra information to the substitutee

Figure 3.7: A simple Generalisation transition/substitution.



term. Such a transition should be scored $0.0$ as `truck` is clearly an unsuitable substitute for `transport`.

However, we do not want to simply score all Generalisation transitions at $0.0$. Once again, we will use the breadth of the transition as a guide for calculating the score. The breadth, in this case, is the total number of Generalisation transitions that links to the entity acting as the substitutee. Note that this is just the breadth of the equivalent Specialisation transition in reverse.

When the breadth is one, we will again give a maximum score of $0.99$: $Gn_{max} = 0.99$. But we are no longer concerned with the minimum score, which will simply be $0.0$. A slightly different equation will be used to calculate the exact score, which magnifies the penalty as the breadth increases (as opposed to the dampening in Specialisation transitions scores). The exponentiation applied to the breadth helps to amplify the undesirability of generalisations/assumptions. We have also added a slight expansion to the key coefficient term of $0.1$. This expansion puts a hard limit on the breadth allowed for Generalisation transitions, effectively clips the score at $0.0$ for any such transition with a breadth higher than four.

$$score = \max(0, Gn_{max} \cdot (\frac{1.1}{breadth^{1.5}} - 0.1)) \tag{3.3}$$

Using Equation (3.3), a transition from `bobcat` to `lynx` will have a score of $0.29$ (2dp), while `truck` to `transport` will score the expected $0.0$.

### 3.1.4 Search

We now have the means to match arbitrary phrases to entities in a knowledge base as well as a method to quantify the information loss when moving between related entities. But what happens when the subsitute targets are some distance away from the substitutee targets (i.e. multiple transitions are required)? Information loss should be carried, or accumulated over every transition moving from the substitute to the substitutee.

We can achieve this effect by taking the product of transition scores along a path from the substitute to the substitutee. So as more information is lost over each transition, the overall information retention decreases, which eventually becomes the final substitutability score. For example, if two transitions both retain half of the original information, then traversing both in sequence would lower the information retention to a quarter.

$$\begin{aligned} \texttt{food} \rightarrow \texttt{dessert} &= 0.5 \\ \texttt{desert} \rightarrow \texttt{cake} &= 0.5 \\ \texttt{food} \rightarrow \texttt{dessert} \rightarrow \texttt{cake} &= 0.25 \end{aligned}$$

All that is left to do is to find a path between the two sets of targets which maximises substitutability, or minimises information loss. To find such a path, we first construct a graph structure where the nodes represent entities and edges represent transitions. Then we employ a best-first search algorithm that is similar to Dijkstra's algorithm [8], but with a preference for the highest scoring nodes instead of lowest path cost (see Figure 3.8). The score of each node is simply the product of all the transitions used to get to it from the initial substitute target.

The search continues until a path reaches the substitutee target. The score of this target node is then returned as the overall substitutability of the path found. If no path can be found, then $0.0$ is returned.

Figure 3.8: A path of maximum substitutability from `food` to `cake`.

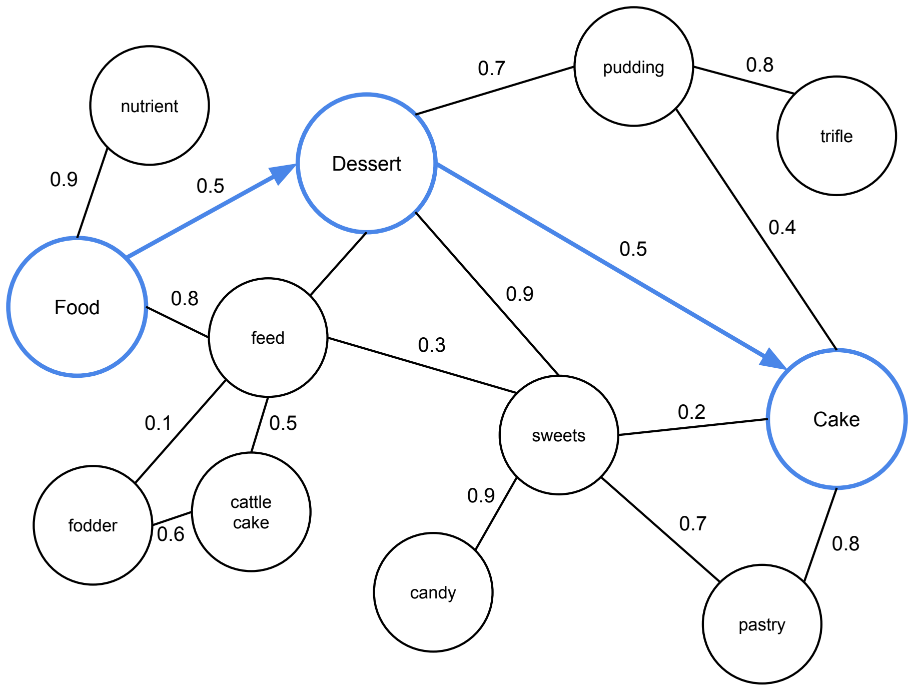
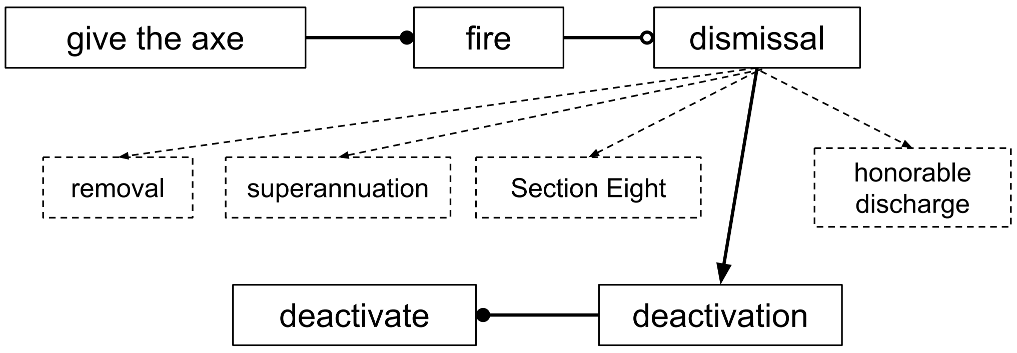


Figure 3.9: A path of maximum substitutability from `give the axe` to `deactivate`.



When multiple targets are present (most of the time), all targets from the substitute's side are initially added to the fringe of the search with a score of $1.0$. Then for every new node that is expanded to, it is checked against all the targets from the substitutee's side. The first path that is found which links any substitute target to any substitutee target will be the final path used.

The act of finding a path of maximum information retention also has the added benefit of disambiguating the substitute and substitutee against each other. As the two targets which are found at the ends of this path should reflect the sense (or form) of the two phrases that put them in as similar a context as possible, thus maximising substitutability.

Figure 3.9 showcases another example of a path of substitutability, but displayed as a series of transitions as described in Section 3.1.3. The final substitutability score of the pair `give the axe` $\rightarrow$ `deactivate` is calculated as follows:

$$\begin{aligned}
\texttt{give the axe} &\rightarrow \texttt{fire} = 0.99\\
\texttt{fire} &\rightarrow \texttt{dismissal} = 0.7\\
\texttt{dismissal} &\rightarrow \texttt{deactivation} = 0.66\\
\texttt{deactivation} &\rightarrow \texttt{deactivate} = 1.0\\
\texttt{give the axe} &\rightarrow \texttt{deactivate} = \mathbf{0.46}
\end{aligned}$$

Algorithm 4 shows exactly how the search is performed given two target sets. The sub-routine EXPAND returns a set of transitions from a given entity, along with its transition score.

**Algorithm 4** Finds the best path of substitutability from the target set $ts1$ to $ts2$. Returns the score/substitutability of the path.

```
    procedure SEARCH($ts1, ts2$)
        $fringe \leftarrow$ max heap
        $visited \leftarrow$ empty set
        for all $t$ in $ts1$ do
 5:         $node \leftarrow$ TONODE($t$)  ▷ converts a target to a node with score $1.0$
            offer $node$ to $fringe$
        end for
        while $fringe$ is not empty do
            $node \leftarrow$ poll from $fringe$
10:         if $node.entity$ already in $visited$ then
                continue
            else if $node.entity$ is in $ts2$ then
                return $node.score$                         ▷ best path found
            end if
15:         add $node.entity$ to $visited$
            if $neigh.score <= 0.0$ then                    ▷ ignore if score is 0
                continue
            end if
            $transitions \leftarrow$ EXPAND($node.entity$)
20:         for all $tr$ in $transitions$ do
                if $tr.next$ not in $visited$ then          ▷ the entity reached by $tr$
                    $neigh \leftarrow$ TONODE($tr.next$)
                    $neigh.score \leftarrow node.score * tr.score$
                    offer $neigh$ to $fringe$
25:             end if
            end for
        end while
        return $0.0$                                        ▷ no valid path found
    end procedure
```

## 3.2 WordNet implementation

WordNet [45] is a free, publicly available database of English words and common phrases. Words in WordNet have a many-to-one correspondence with words in a traditional dictionary (i.e. the lexical, or string-of-letters, representation of words). This is because every usage, meaning and context of each lexical word is its own individual WordNet word. In this section, "word" will refer specifically to a WordNet word. For example, the lexical word `complete` exists as 10 individual words in WordNet – five verbs and five adjectives.

- Verb
  - come to a finish or end
  - bring to a whole, with all necessary parts
  - carry out
  - complete a pass
  - write all required information onto a form
- Adjective
  - having all the normal parts, components or steps
  - perfect, having all qualities
  - accomplished
  - pejorative intensifier
  - been brought to a conclusion

Words are grouped together in sets of tightly-coupling synonyms, called **synsets**. Each word can be in more than one synset. Each synset expresses a distinct concept, whereby all the words in the synset can be used in place of one another within the context of that concept. Therefore, synsets are collections of words that have perfect substitutability within the context of each other (i.e. if disambiguited against each other). Figure 3.10 lists the synsets for the word `complete` using WordNet's web interface (`http://wordnetweb.princeton.edu/perl/webwn`). Note that each synset does not necessarily represent a different meaning for `complete`. For example, consider the synsets `[complete, finish]` and `[complete, fill out, fill in]`. The meaning/definition of `complete` is almost identical in both synsets, but the usage and context is different. Synsets are also interlinked by means of semantic and lexical relationships. This network structure created by WordNet makes it a useful tool for natural language processing and a perfect knowledge base to mine for our WordNet-based implementation of a substitutability system – **WordSub**.

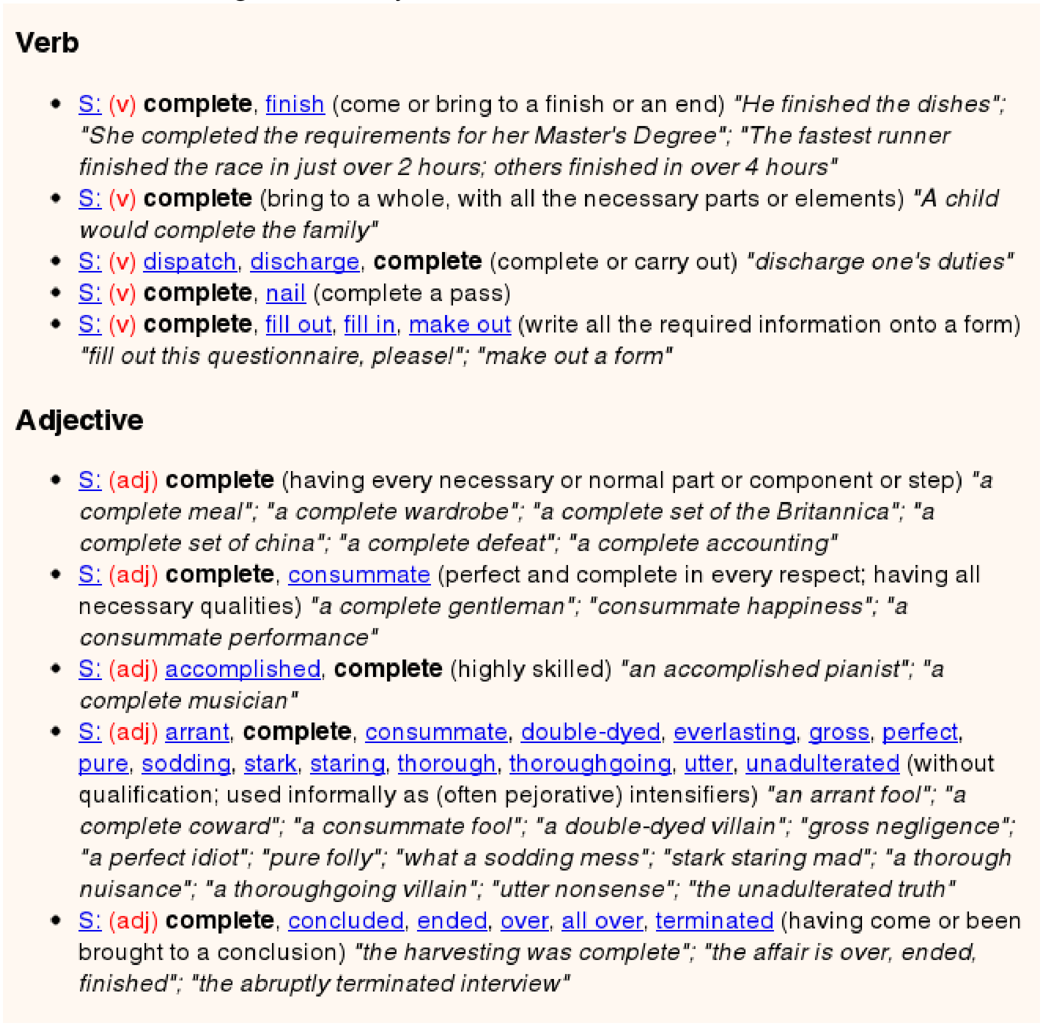
To assist WordSub in the task of mining WordNet, we will be using JWI the MIT Java WordNet Interface [14]. The library supports the latest versions of WordNet and provides an easy to use Java API. JWI accesses WordNet locally via an offline database (known as a dictionary). We obtained the latest WordNet dictionary available at the time – version 3.1, last updated in June 2011.

### 3.2.1 Targets

The first task that WordSub has to perform is to match the plain text input terms (the substitute and the substitutee) to "entities" in WordNet. Words are the most basic elements of WordNet, which make them a great starting point for creating target sets. So to begin, WordSub uses JWI to match the input terms to words. If a match cannot be found directly, then it will be broken down as shown in Algorithm 2. After this process completes, WordSub is left with two sets of WordNet words.

However, there is a slight problem with using words as targets. Because words in WordNet do not represent distinct concepts on their own, they are only linked by lexical relationships (e.g. grammatical derivations

Figure 3.10: Synsets for the word `complete`.



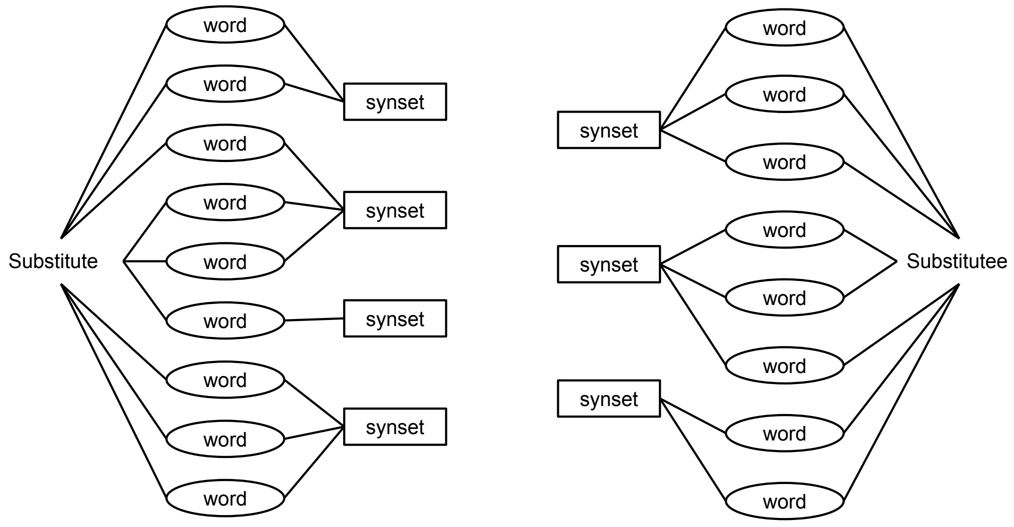
**Verb**

- S: (v) **complete**, finish (come or bring to a finish or an end) *"He finished the dishes"; "She completed the requirements for her Master's Degree"; "The fastest runner finished the race in just over 2 hours; others finished in over 4 hours"*
- S: (v) **complete** (bring to a whole, with all the necessary parts or elements) *"A child would complete the family"*
- S: (v) dispatch, discharge, **complete** (complete or carry out) *"discharge one's duties"*
- S: (v) **complete**, nail (complete a pass)
- S: (v) **complete**, fill out, fill in, make out (write all the required information onto a form) *"fill out this questionnaire, please!"; "make out a form"*

**Adjective**

- S: (adj) **complete** (having every necessary or normal part or component or step) *"a complete meal"; "a complete wardrobe"; "a complete set of the Britannica"; "a complete set of china"; "a complete defeat"; "a complete accounting"*
- S: (adj) **complete**, consummate (perfect and complete in every respect; having all necessary qualities) *"a complete gentleman"; "consummate happiness"; "a consummate performance"*
- S: (adj) accomplished, **complete** (highly skilled) *"an accomplished pianist"; "a complete musician"*
- S: (adj) arrant, **complete**, consummate, double-dyed, everlasting, gross, perfect, pure, sodding, stark, staring, thorough, thoroughgoing, utter, unadulterated (without qualification; used informally as (often pejorative) intensifiers) *"an arrant fool"; "a complete coward"; "a consummate fool"; "a double-dyed villain"; "gross negligence"; "a perfect idiot"; "pure folly"; "what a sodding mess"; "stark staring mad"; "a thorough nuisance"; "a thoroughgoing villain"; "utter nonsense"; "the unadulterated truth"*
- S: (adj) **complete**, concluded, ended, over, all over, terminated (having come or been brought to a conclusion) *"the harvesting was complete"; "the affair is over, ended, finished"; "the abruptly terminated interview"*

Figure 3.11: Finding target sets in WordNet.



from one word to another). For example, the word `compute` can only be linked to its other forms, such as `computation`, `computational` and `computer`. This essentially has the same effects as simply comparing the stems. WordSub is more interesting when exploiting the semantic relationships that exist between synsets. Therefore, it is more beneficial to treat entire synsets are targets and entities, even though they are not the most basic units of knowledge in WordNet. After generating the two sets of WordNet words, the synset of each word is then retrieved using JWI to create the target sets. Since there is a many-to-many relationship between words and synsets, WordSub ensures duplicate synsets are removed. Figure 3.11 illustrates the process of generating target sets.

### 3.2.2 Transitions

Now that WordSub has the means to produce the target sets of synsets, the next step is to find a path between the substitute and substitutee, by transitioning from one synset to another. Luckily, WordNet boasts a large number of different types of relationships between synsets (and also words), of which the version of JWI WordSub uses has access to 27. By examining a subset of these relationships (a.k.a. *pointers* in JWI), WordSub is able to fully implement all four types of substitution transitions as defined in Section 3.1.3 – **Same**, **Similar**, **Specialisation** and **Generalisation**.

### Same

Same transitions represent a perfect or near-perfect substitution between two entities. Since synsets represent distinct concepts, it is unlikely that a semantic transition between any two synsets will be perfect. However, as previously stated, the words within any one synset can be freely used in place of one another in the context of the concept represented by that synset. Therefore we reason that by using synsets as entities instead of words, Same transitions are implemented implicitly by WordSub. For example, the terms `temper` and `harden` have the same meaning in the context of metal and glass work. As a result, they will generate target sets that share a common synset, making the two terms perfectly substitutable.

### Similar

Similar transitions represent good, but imperfect substitution between two entities. WordSub will explore both lexical and semantic links between synsets when implementing Similar transitions. For the latter, two semantic pointers will be considered – *similar to* and *see also*. Both pointers link synsets which are similar in meaning/definition, i.e. they are rough synonyms which cannot be used perfectly in place of one another. For example, the synset `[nasty, awful]` has the following *similar to* and *see also* pointers, all of which can be used to implement a Similar transition. WordSub scores these transitions at $0.5$, which falls within the Similar transition score range of $0.5 - 0.8$.

- see also
    - `[unpleasant]`
- similar to
    - `[dirty, filthy, lousy]`
    - `[grotty]`
    - `[hateful, mean]`

WordSub also explores grammatical derivations for purpose of Similar transitions. Recall that grammatical/lexical derivations are actually considered to be a Similar transition unless used to link directly from/to entities within target sets (see Section 3.1.3). WordNet supplies three bidirectional pointer types for grammatical derivations – noun-adjective, noun-verb and verb-adjective. The problem is, grammatical derivations only exist between words, not synsets, as they are considered lexical relationships. WordSub solves this issue by making the following assumption: if word `x` is in synset `A` and word `y` is in synset `B`, and there exists a derivation pointer from `x` to `y`, then there is also an implied derivation from synset `A` to `B`. WordSub scores derivation transitions at 0.7. For example, the synset `[clear, decipherable, readable]` can be linked to the following synsets via derivation pointers:

- `[clarity, lucidity, clearness, limpidity]`
- `[clearness, clarity, uncloudedness]`
- `[readability]`
- `[legibility, readability]`

Note that *similar to* and *see also* pointers only exist for adjective synsets, while derivation pointers exist for nouns, verbs and adjectives.

### Specialisation and Generalisation

To move in directions of generality and specificity, WordSub exploits the super-subordinate relationships among synsets. The most common super-subordinate relationship exists between noun synsets. By following *hypernym* pointers, WordSub can easily transition from one synset to its more generic counterpart. For example, there exists a hypernym pointer from `[desk]` to `[table]`. Similarly, WordSub can move in the direction of specificity by following *hyponym* pointers. For example, there exists a hyponym pointer from `[rain, rainfall]` to `[drizzle, mizzle]`. Super-subordinate relationships between noun synsets form a connected tree structure, rooted at the synset `[entity]`. This means WordSub can reach any noun synset in WordNet by following hyper/hyponym pointers. WordNet also differentiates between relationships of *type* (i.e. is-a-type-of) and *instance* (i.e. is-a). For example, `[river]` is a hyponym of `[stream, watercourse]`, but `[Nile, Nile River]` is a hyponym *instance* of `[river]`, which also makes it a leaf in the overall tree network. In practice, distinguishing between *type* and *instance* (i.e. giving different transition scores) has little effect on the final outcome, so we treat them as if they were the same.

Verb synsets are also organised into a hierarchical structure. WordSub can still move in the direction of generality by following hypernym pointers. But to move towards specificity, *troponym* pointers are used instead. A troponym is a verb that indicates a more precise (or specific) manner of doing something. For example, there exists a troponym pointer from `[run]` to `[sprint]`. Unlike noun synsets, verb synsets do not share a common root, resulting in a forest network instead of a single connected tree. Consequently, WordSub is limited in its transitions when dealing with verbs, relying more on Similar transitions to reach more *distant* synsets.

The breadth of all these Specialisation and Generalisation transitions is the branching factor of the pointer being followed. In the case of noun synsets, the breadth is the number of hyponyms (type or instance) a synset has. In the case of verb synsets, the breadth is the number of troponyms a synset has. WordSub will then calculate transition scores as defined by Equations (3.2) and (3.3).

### 3.2.3 Examples

This section will run through some examples of subsitution paths that WordSub would produce.

For the first example, WordSub is given the input terms `admire` and `appreciation`. That is, WordSub needs to calculate the substitutability of using `admire` in place of `appreciation`. Table 3.1 shows the initial target sets which WordSub will generate from the inputs. Since there is not an overlap between the two sets, WordSub now has to find a subsitution path between the two. During this search, WordSub will discover that the word `admire` from the synset `[admire, look up to]` has a derivation pointer to the word `admiration`. `Admiration` can be found in the synset `[admiration, appreciation]`, thus completing the substitution path (see Figure 3.12). The final score of this substitution is therefore $0.7$, taken from the single derivation transition.

Table 3.1: Target sets for `admire` and `appreciation`, generated by WordSub.

| **admire** | **appreciation** |
|---|---|
| [admire, look up to] | [appreciation, grasp, hold] |
| [admire] | [taste, appreciation, discernment, perceptiveness] |
| | [appreciation (expression of gratitude)] |
| | [admiration, appreciation] |
| | [appreciation (in value)] |

Figure 3.12: Simple subsitution from `admire` to `appreciation` found by WordSub.

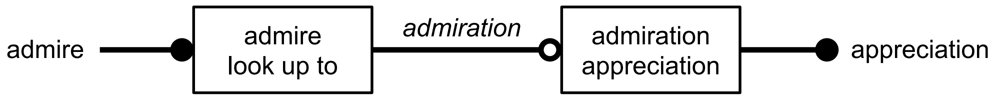
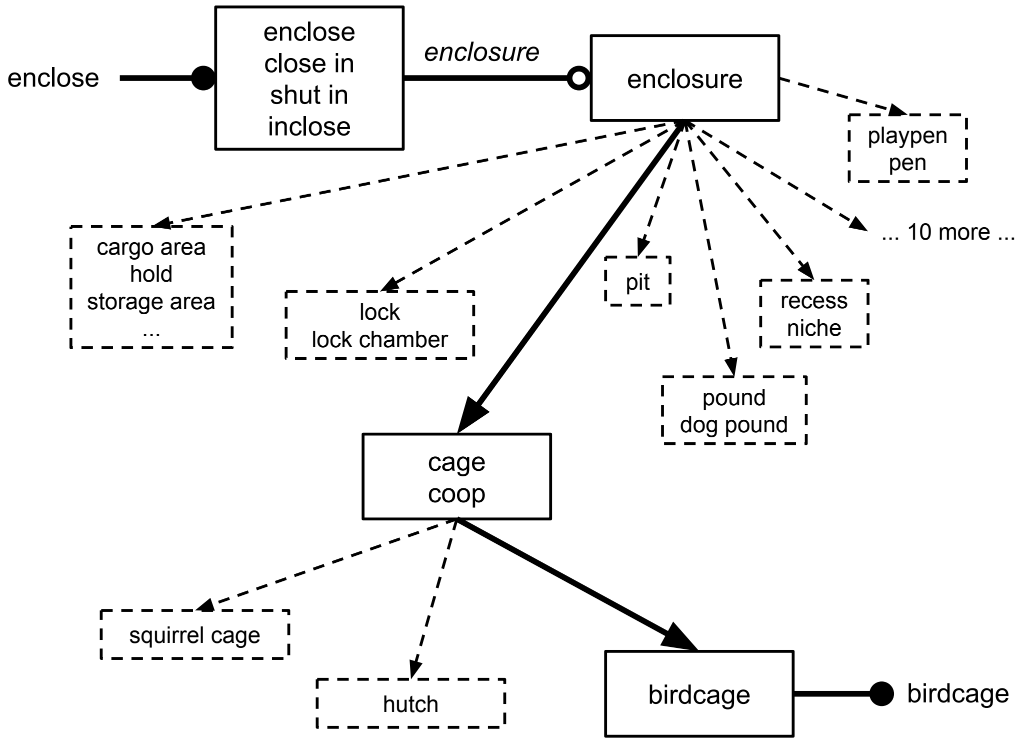


Table 3.2: Target sets for `enclose` and `birdcage`, generated by WordSub.

| **enclose** | **birdcage** |
|---|---|
| [envelop, enfold, enwrap, wrap, enclose]<br>[enclose, hold in, confine]<br>[enclose, close in, inclose, shut in]<br>[insert, enclose, inclose, stick in, put in, introduce] | [birdcage] |

The next example is more complicated. We will see how WordSub calculates the substitutability of using `enclose` in place of `birdcage`. First, the two target sets are generated as shown in Table 3.2. Once again, there is no overlap between the sets, so a search will commence. As WordSub explores the network of synsets, it will eventually find that the word `enclose` in the synset `[enclose, close in, inclose, shut in]` has a derivation pointer to the word `enclosure`, which can be found in its own synset. WordSub can then transition along a hyponym pointer from `[enclosure]` to `[cage, coop]`. Continuing towards specificity, `[birdcage]` can be reached via a further hyponym pointer, thus completing the substitution path as shown in Figure 3.13.

Figure 3.13: Complex subsitution path from `enclose` to `birdcage` found by WordSub.



The final substitutability score of `enclose` $\rightarrow$ `birdcage` is calculated as follows:

$$\begin{aligned}
\texttt{enclose} &\rightarrow \texttt{[enclose, close in, ...]} = 1.0 \\
\texttt{[enclose, close in, ...]} &\rightarrow \texttt{[enclosure]} = 0.7 \\
\texttt{[enclosure]} &\rightarrow \texttt{[cage, coop]} = 0.51 \\
\texttt{[cage, coop]} &\rightarrow \texttt{[birdcage]} = 0.75 \\
\texttt{enclose} &\rightarrow \texttt{birdcage} = \mathbf{0.27}
\end{aligned}$$

## 3.3 Wikipedia implementation

Wikipedia (`https://www.wikipedia.org/`) is a freely available, online encyclopedia of interlinked articles. It represents a vast multilingual knowledge base of concepts (i.e a particular topic or entity) and the semantic relationships between concepts. This makes it an ideal resource for natural language processing if one can effectively extract or mine semantic information from its large database. We will be mining the semantic-rich information contained in this knowledge base for our Wikipedia-based implementation of a substitutability system – **WikiSub**.

Full data dumps of Wikipedia are available in the form of large XML and HTML files, hosted by the Wikimedia Foundation [25]. Mining from all this data is a slow and tedious process, because the files are enormous in size and the markup is not friendly to parse into anything useful – making it far too inefficient for our purposes. Luckily in 2012, the University of Waikato released an open-source toolkit called Wikipedia Miner [46], which can be used to perform this task much more efficiently.

We obtained a full data dump of Wikipedia of July 2011 as a single XML file approximately 30 gigabytes in size. Wikipedia Miner was then used to extract and parse the file into a high performance database, based on the Berkeley database engine [49]. Wikipedia Miner then provides a Java API to efficiently access the database through a set of predefined models.

### 3.3.1 Targets

Besides providing an efficient method of accessing Wikipedia, Wikipedia Miner also includes a set of other useful features, such as comparing the relatedness of articles and disambiguating links. The main feature that interested us is the ability to match plain text input to Wikipedia *labels*. A *label* is a model used by Wikipedia Miner to represent an anchor text (clickable text in a hyperlink) that links to specific articles or pages in Wikipedia. Although each label has a specific term/phrase attached to it, most are still ambiguous. In other words, labels do not represent individual entities in Wikipedia.

Instead, each label links to many articles or concepts, which are the actual knowledge entities in Wikipedia. Given a label, Wikipedia Miner is able to return the list of articles that it links to – known as the *senses* of a label. For example, the label `red planet` will link to the following articles:

- Red Planet (novel) – by Robert A. Heinlein
- Red Planet (miniseries) – a 1994 adaptation of the novel
- Mars – the 4th planet from the Sun
- Red Planet (film) – a 2000 film starring Val Kilmer
- Red Planet – a song by Little Mix

We retrieve the senses for both input phrases (substitute and substitutee), which are then used as target sets in WikiSub. If a label cannot be matched to one or both of the input phrases, then it will be broken down into its component aspects until a match can be found as per Algorithm 2. In WikiSub, the sub-routine MATCHTARGETS will return a set of senses if a label can be matched to the input. The two procedures, FINDTARGETS from Algorithm 1 and FINDASPECTS from Algorithm 2, both return lists of such sets.

There is just one problem with using label senses. Some labels are resolved to a huge number of senses, most of which will be irrelevant to the label itself. A sense will be linked to a label if even a single anchor text exists (with that label's text) that links to the sense. WikiSub requires a method for determining how *common* a sense is in the context of a particular label. Uncommon senses can then be pruned out of the target set returned by MATCHTARGETS. Luckily, Wikipedia Miner provides a summary statistic which is perfect for this task – a sense's prior probability.

The prior probability of a sense measures the probability that a particular label actually links to that sense. Put simply, it is the fraction of anchor texts which link directly to that sense over the total number of anchor texts for the label in question.

WikiSub removes a label's irrelevant senses by accepting them in descending order of prior probability and keeping a running total of all the priors. Once the total prior exceeds a certain threshold, which we call the maximum prior coverage, the remaining senses are rejected/pruned. WikiSub uses a maximum prior coverage value of $0.99$, which means the accepted senses cover $99\%$ of all anchor texts within Wikipedia that are tied to the current label being processed. To further ensure irrelevant senses are eliminated, WikiSub also rejects any senses with a prior probability less than $0.001$.

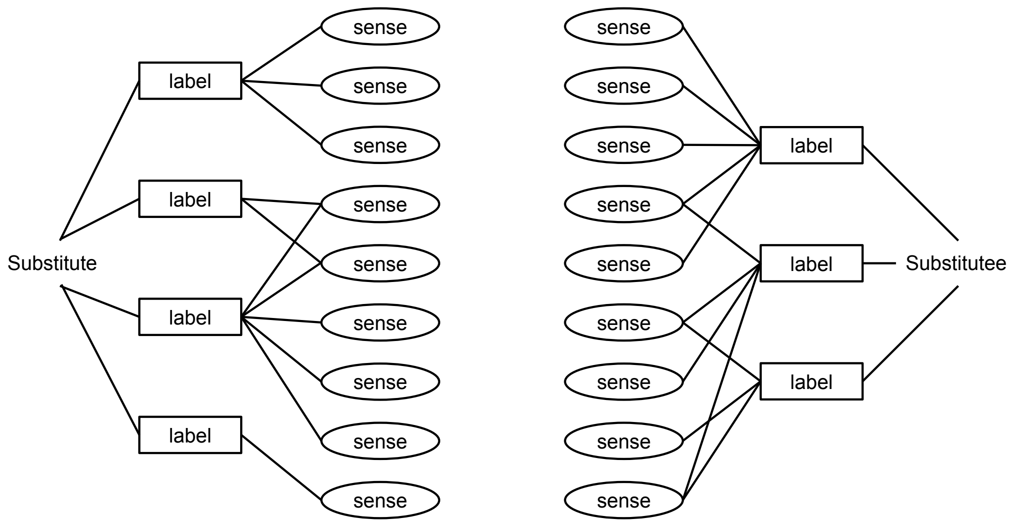
### 3.3.2 Transitions

By following the semantic links between the senses extracted by Wikipedia Miner, WikiSub is able to implement three out of the four transitions defined in Section 3.1.3 – **Same**, **Specialisation** and **Generalisation**. We explored the idea of following links in the "See also" section of Wikipedia articles for **Similar** transitions, but these were not a reliable source as they often included links to articles that were completely unrelated from the perspective of substitutability.

#### Same

Recall that one form of Same transitions occur between entities that are grammatical inflections of each other. We implemented these transitions by first stemming the plain text input (using the Porter stemming algorithm [54]) prior to matching it to Wikipedia labels. Therefore, all labels that matched the stem of our input would be retrieved. Since this form of the Same transition had a perfect transition score of $1.0$, there is no harm

Figure 3.14: Finding target sets in Wikipedia.



in simply putting the senses of all matching labels into the initial target set (remove any duplicates caused by overlaps in senses), which is what WikiSub does. The initial process of generating the target sets is represented graphically in Figure 3.14.

The other form of Same transitions occur between entities with the same meaning but are lexically different. One way of implementing these transitions is by examining *redirect* links that feed into each article/sense. *Redirects* are Wikipedia pages that solely connect an article to its common alternative titles. Such alternatives usually correspond to synonyms – `Baby cat` redirects to `Kitten`; typical spelling variations – `Liquorice` redirects to `Licorice`; and acronym expansions – `AI` redirects to `Artificial intelligence`.

Another way WikiSub implements Same transitions is by exploiting Wikipedia *categories*. Most (almost all) Wikipedia articles belong to one or more categories, which group a set of similar topics and concepts together. Many categories also have a central article that is its equivalent. For example, the article `Computer science` belongs in the category `Computer` `science` and is also its central article. This occurs when the topic within a particular article is broad enough to warrant its own category.

To determine whether an article/sense is central to its parent category, WikiSub checks if the title of the category matches that of the article or any alternative titles (redirects) of that article, which covers most cases. Occasionally, a category's central article will have a name with a different inflection or have an additional bracketed term at the end of the category title for disambiguation purposes. Therefore, we strip any bracketed terms and take the stem of an article's title when checking whether it is central to a category or not. For example, the article `Orange (fruit)` belongs to the following categories:

- Citrus hybrids
- Cocktail garnishes
- Oranges
- Symbols of California
- Symbols of Florida
- Tropical agriculture

WikiSub will recognise that the article `Orange (fruit)` is in fact the central article in the category `Oranges`, and thus create a Same transition between the two. By allowing such transitions, WikiSub is able to expand its search to articles, redirects and categories.

### Specialisation and Generalisation

Now that we have the means to move from a sense to a category, WikiSub can exploit the hierarchical organisation of categories in order to transit to and from topics that are either more general or specific. All Wikipedia categories descend from a single root category [46], the category `Contents`.

So when WikiSub transits to a category, it then has access to what is essentially a taxonomy representing all of human knowledge – organised in order of generality. Consequently, it will also be able to reach almost all other articles within Wikipedia.

There are two ways that WikiSub implements Specialisation transitions. The first is by moving from a category to one of its child categories or sub-categories. The breadth of this transition is then the number of total sub-categories (see Section 3.1.3 for how breadth is used to calculate transition scores). For example, `Lakes` is a sub-category of `Bodies of water`, which makes `Bodies of water` $\rightarrow$ `Lakes` a Specialisation transition. The second way is to move from a category to one of its non-central child articles (recall that moving from a category to its central article is handled as a Same transition). The breadth of this transition is then the number of total non-central child articles. For example, `Latte` is a child article of `Coffee beverage`, which makes `Coffee beverage` $\rightarrow$ `Latte` a Specialisation transition.

Similarly, WikiSub implements the Generalisation transition in the same two ways, but in reverse order. One is by moving from a category to its parent category. Note that because Wikipedia categories often have more than one parent category, the breadth of such a transition could either be the total number of parent categories or the total number of sub-categories belonging to the parent category. We decided to stick with the latter option as it conforms to the definitions discussed in Section 3.1.3. Another way of obtaining a Generalisation transition is to move from an article to one of its parent categories, as long as that article is non-central to the parent category. Again, the breadth of the transition is the total number of non-central articles in that parent category.

### 3.3.3 Examples

This subsection will run through some examples of subsitution paths that WikiSub would produce.

For the first example, the inputs to WikiSub are the terms `Pennsylvania` and `Keystone state`. That is, we would like WikiSub to measure the substitutability of using `Pennsylvania` in place of `Keystone State`. `Pennsylvania` produces a large number of senses/targets, which include the following:

- University of Pennsylvania
- Pennsylvania
- USS Pennsylvania (1837)
- Pennsylvania (steamboat)

`Keystone State` only produces three senses/targets as follows. One of the targets (highlighted in bold) overlaps with one of the targets from `Pennsylvania`.

- USS Keystone State (1853)
- **Pennsylvania**
- List of U.S. state nicknames

Since the two target sets already overlap, WikiSub does not need to perform a search between the sets and the output substitution path will simply be the single node representing the article `Pennsylvania` (see Figure 3.15). The final substitution score is a perfect $1.0$.

The second example is less trivial. WikiSub will attempt to calculate the substitutability of using `natural science` in place of `organic chemistry`. The initial set of generating target sets will produce the senses

Figure 3.15: Simple subsitution from `Pennsylvania` to `Keystone State` found by WikiSub.

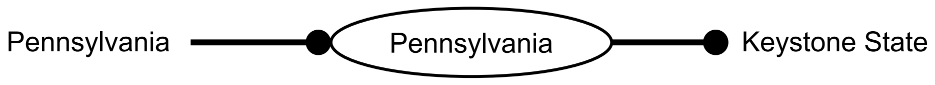


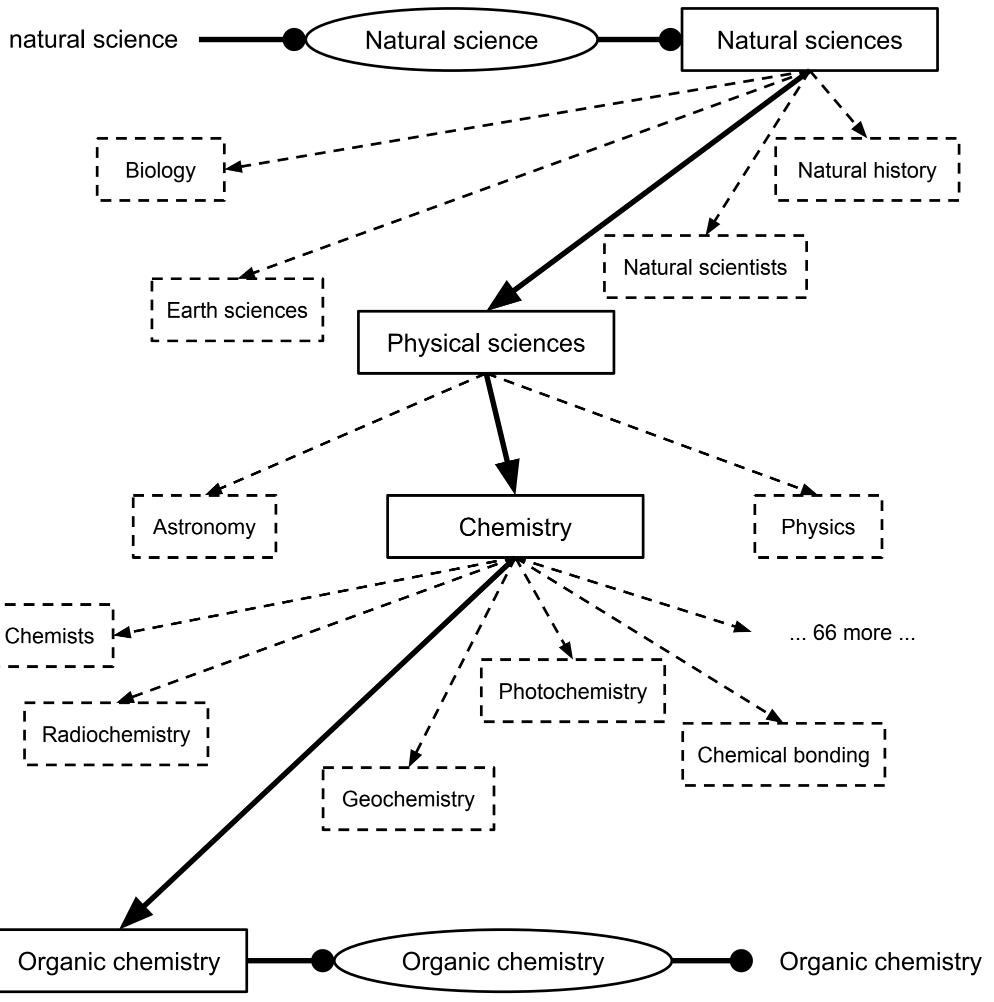
Table 3.3: Target sets for `natural science` and `organic chemstry`, generated by WikiSub

| **natural science** | **organic chemstry** |
|---|---|
| Science | Organic chemistry |
| Natural science | |
| Natural Sciences (Cambridge) | |
| Science in medieval Islam | |
| Natural Sciences (Durham) | |

shown in Table 3.3. Since there is no overlap between the two target sets, WikiSub will attempt to make a connection between the two by searching for a substitution path.

During this search, WikiSub will find that the article `Natural science` is the central article for the category `Natural sciences`, and will therefore create a Same transition between the two. From there, WikiSub can reach the `Physical sciences` category (a sub-category of `Natural sciences`) via a Specialisation transition. Continuing on this path of specificity, WikiSub will expand its search to the sub-category `Chemistry` and then `Organic chemistry`. Finally, it will see that the category `Organic chemistry` has a central article with the same name, which also happens to match the sense in the substitutee's target set, thus completing the substitution path. Figure 3.16 shows the graphical representation of this path – rectangles represent categories while ovals represent senses/articles.

Figure 3.16: Complex subsitution path from `natural science` to `organic chemistry` found by WikiSub.



The final substitutability score of `natural science` → `organic chemistry` is calculated as follows. Terms prefixed with "`C:`" represent categories.

$$\begin{aligned}
\texttt{Natural science} &\rightarrow \texttt{C:Natural sciences} = 1.0\\
\texttt{C:Natural sciences} &\rightarrow \texttt{C:Physical sciences} = 0.66\\
\texttt{C:Physical sciences} &\rightarrow \texttt{C:Chemistry} = 0.75\\
\texttt{C:Chemistry} &\rightarrow \texttt{C:Organic chemistry} = 0.39\\
\texttt{C:Organic chemistry} &\rightarrow \texttt{Organic chemistry} = 1.0\\
\texttt{Natural science} &\rightarrow \texttt{Organic chemistry} = \mathbf{0.19}
\end{aligned}$$

# Chapter 4

# Evaluation

In this chapter, we focus on the task of measuring the performance of substitutability systems. First, we describe the process of how we created a dataset for the specific purpose of evaluating substitutability systems is explained. Second, a variety of different evaluation metrics are introduced, with the motivation behind each one justified.

## 4.1 Dataset

We want a dataset that can test the performance of our substitutability system against existing approximate string matching systems. This dataset should contain both substitution pairs that are lexically similar, suitable for non-semantic partial string matching; and pairs that are only semantically substitutable, suitable for knowledge based systems such as ours.

There are existing datasets available for evaluating systems that perform only approximate lexical string matching. However, none of these are suitable for purely evaluating systems that measure substitutability. More specifically, they do not test for semantic substitutability. For example, in [73] and [30], the dataset consisted of substitute-substitutee pairs that were only substrings of each other. In reality (i.e. when used to evaluate the performance of a keyphrase extraction system against a set of gold

Table 4.1: Example question given to volunteers for dataset creation.

| **Substitutee:** | APPLES | | | |
|---|---|---|---|---|
| **Substitute:** | Fruit | Pear | Vegetables | Food |

standard keyphrases, see Chapter 1), approximate string matching systems have to cope with any arbitrary pair of phrases. More semantically-oriented datasets are also available, such as the WordSimilarity-353 Test Collection [13] – a set of English word pairs with human-assigned similarity judgements. But these are focused around the notion of semantic relatedness, which can differ significantly to substitutability in many cases (see Section 2.1). Therefore, we decided to create our own dataset, in order to isolate the task of measuring substitutability.

With the help of over 130 volunteers (university students ranging from undergraduate to doctorate), we created a dataset which consisted of 88 *questions*. Each question targeted a single substitutee with four potential substitutes, as shown in Table 4.1.

For each question, the volunteers were asked to circle the best substitute out of the four and cross out any that are definitely unsuitable as substitutes (i.e. a substitutability of $0.0$ or near $0.0$). They were also given the option to circle up to two substitutes if they could not decide which was better. We believe that making the volunteers compare a set of substitutes against a single substitutee produces more consistent and reliable data, as opposed to asking them to assign arbitrary scores to independent substitute-substitutee pairs. For example, it is easy to agree that `Fruit` is a better substitute than `Vegetables` for `Apple`, but assigning a specific score for the pair `Fruit` $\rightarrow$ `Apple` is both more difficult and more subjective. To ensure there were no guesses, volunteers were told to ignore any questions where they did not know the meaning of all five phrases – the substitutee and four substitutes. Note that the volunteers were not given the strict definition of substitutability laid out in Section 3.1.1, as we did not want to bias their choices towards our expectations. Instead, they were simply told that substitutability is a measure of how suitable a phrase is when used in place of another.

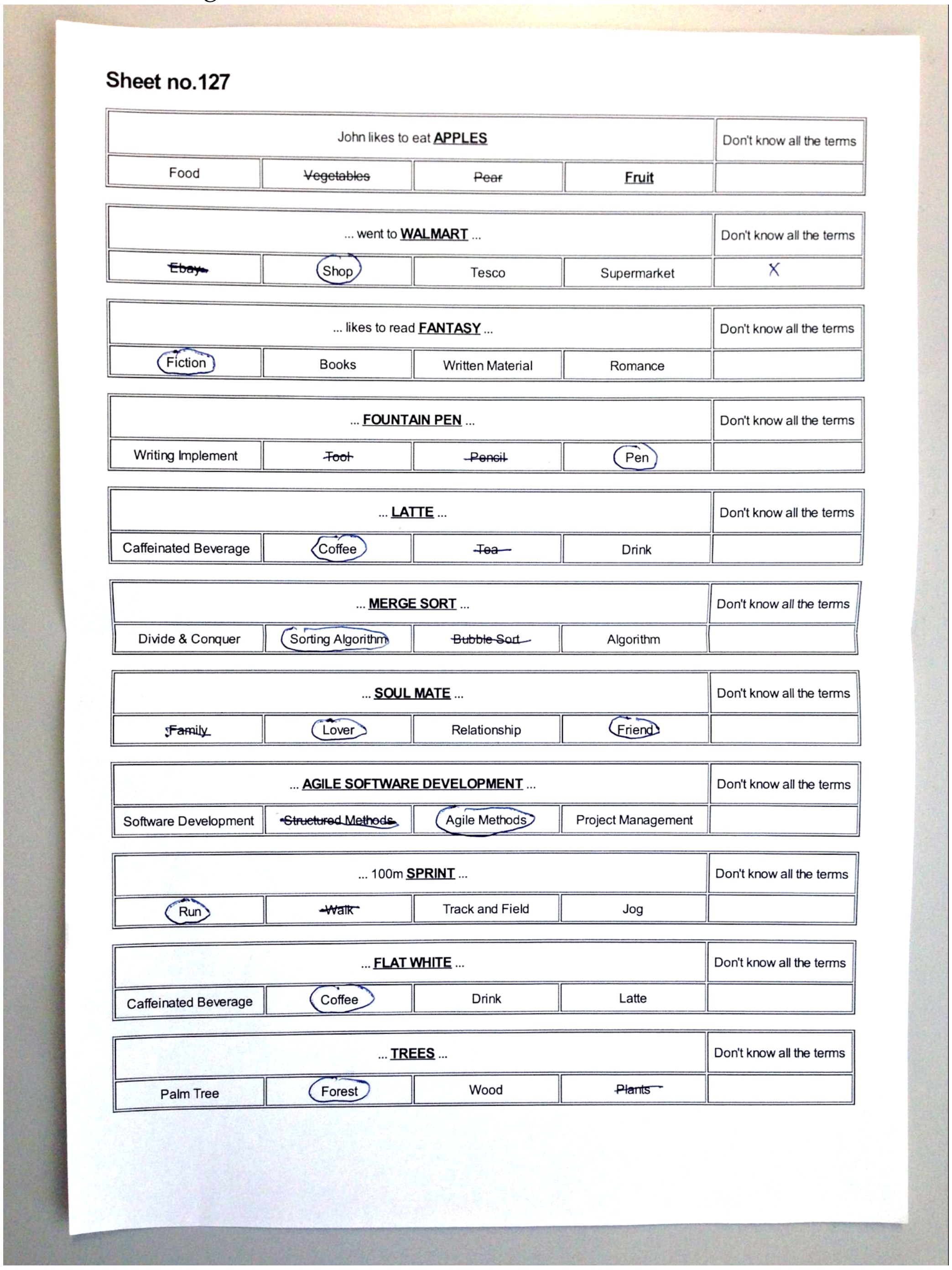
For the example in Table 4.1, a sensible answer would have `Fruit` circled, while crossing out `Vegetables` and `Pear`. Recall that our definition of substitutability – a measure of how much information/meaning is retained if one phrase (substitute) is used in place of another (substitutee), whilst making as few assumptions towards the substitutee as possible (see Section 3.1.1 for a more detailed explanation). `Fruit` retains more information towards `Apple` than `Food`, which is far too general.

Questions were divided into sheets of 10 questions each, along with an additional completed example question at the top. Each sheet contained a unique set of questions (no question appeared twice on a single sheet) and was numbered to assist in data entry later on. Furthermore, the order of the 4 substitutes were also shuffled, to ensure there was no bias caused by the order in which the substitutes were presented to the volunteers. Each volunteer is then handed a single sheet to be completed. Figure 4.1 displays one of these question sheets.

After collecting over 130 sheets, we wrote a simple Java program to assist in the monumental task of entering the data into the computer. When each sheet of 10 questions was initially generated, a data file was saved to track which questions were on it. By entering the sheet number into our data entry program, it read from this data file and immediately present us with the expected list of questions, along with an interface for entering how each question was answered (see Figure 4.2).

A total of approximately 1,200 questions were completed by the volunteers, spread out across all 88 questions (on average, each question was completed over 13 times). The answers were then aggregated to form a single dataset. Each question in fact consists of four substitute-substitutee pairs, but with the same substitutee. A volunteer score is assigned to each of the four substitutes to indicate its substitutability for the substitutee. Every time a substitute was circled by a volunteer, its score was incremented

Figure 4.1: Question sheet for dataset creation.



**Sheet no.127**

| John likes to eat **APPLES** | | | | Don't know all the terms |
|---|---|---|---|---|
| Food | ~~Vegetables~~ | ~~Pear~~ | Fruit | |

| ... went to **WALMART** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| ~~Ebay~~ | [✓ Shop] | Tesco | Supermarket | X |

| ... likes to read **FANTASY** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| [✓ Fiction] | Books | Written Material | Romance | |

| ... **FOUNTAIN PEN** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Writing Implement | ~~Tool~~ | ~~Pencil~~ | [✓ Pen] | |

| ... **LATTE** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Caffeinated Beverage | [✓ Coffee] | ~~Tea~~ | Drink | |

| ... **MERGE SORT** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Divide & Conquer | [✓ Sorting Algorithm] | ~~Bubble Sort~~ | Algorithm | |

| ... **SOUL MATE** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| ~~Family~~ | [✓ Lover] | Relationship | [✓ Friend] | |

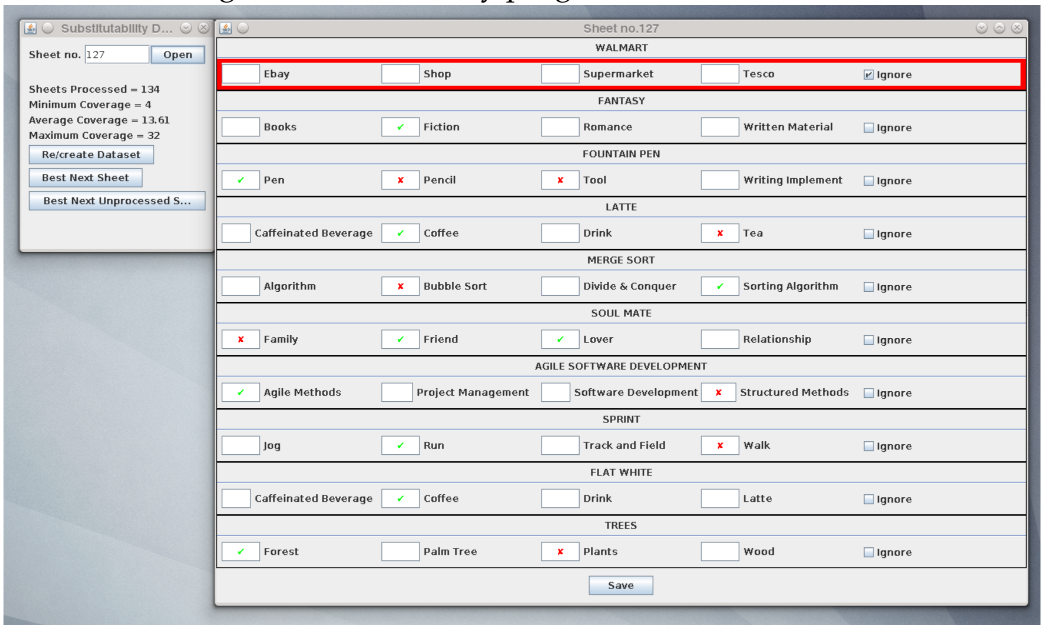
| ... **AGILE SOFTWARE DEVELOPMENT** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Software Development | ~~Structured Methods~~ | [✓ Agile Methods] | Project Management | |

| ... 100m **SPRINT** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| [✓ Run] | ~~Walk~~ | Track and Field | Jog | |

| ... **FLAT WHITE** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Caffeinated Beverage | [✓ Coffee] | Drink | Latte | |

| ... **TREES** ... | | | | Don't know all the terms |
|---|---|---|---|---|
| Palm Tree | [✓ Forest] | Wood | ~~Plants~~ | |

Figure 4.2: Data entry program written in Java.



Substitutability D...

Sheet no. 127 Open

Sheets Processed = 134
Minimum Coverage = 4
Average Coverage = 13.61
Maximum Coverage = 32

Sheet no.127

| | | | | |
|---|---|---|---|---|
| **WALMART** | | | | |
| Ebay | Shop | Supermarket | Tesco | ✓ Ignore |
| **FANTASY** | | | | |
| Books | ✓ Fiction | Romance | Written Material | Ignore |
| **FOUNTAIN PEN** | | | | |
| ✓ Pen | ✗ Pencil | ✗ Tool | Writing Implement | Ignore |
| **LATTE** | | | | |
| Caffeinated Beverage | ✓ Coffee | Drink | ✗ Tea | Ignore |
| **MERGE SORT** | | | | |
| Algorithm | ✗ Bubble Sort | Divide & Conquer | ✓ Sorting Algorithm | Ignore |
| **SOUL MATE** | | | | |
| ✗ Family | ✓ Friend | ✓ Lover | Relationship | Ignore |
| **AGILE SOFTWARE DEVELOPMENT** | | | | |
| ✓ Agile Methods | Project Management | Software Development | ✗ Structured Methods | Ignore |
| **SPRINT** | | | | |
| Jog | ✓ Run | Track and Field | ✗ Walk | Ignore |
| **FLAT WHITE** | | | | |
| Caffeinated Beverage | ✓ Coffee | Drink | Latte | Ignore |
| **TREES** | | | | |
| ✓ Forest | Palm Tree | ✗ Plants | Wood | Ignore |

Table 4.2: Aggregated volunteer scores for one question in the dataset.

| **Substitutee:** ALTERNATING CURRENT (coverage: 15) | | | | |
|---|---|---|---|---|
| **Substitute:** | Electricity | AC | DC | Energy |
| **Score:** | 0 | +15 | -9 | -1 |

by one. Every time a substitute was crossed out by a volunteer, its score was decremented by one. For the example in Table 4.1, `Fruit` would receive $+1$ whilst `Pear` and `Vegetables` would both receive $-1$. Table 4.2 is the aggregated result of one of the 88 questions:

Each question also records the total number of volunteers that answered it (known as its **coverage**), which is used later in the various evaluation metrics (see Section 4.2). The coverage for the example in Table 4.2 is 15, meaning everyone agreed that `AC` (with a maximum volunteer score of 15 also) is the best substitute for `Alternating Current`, while most agreed that `DC` is not a suitable substitute.

Table 4.3: Expected system scores for one question in the dataset.

| **Substitutee:** | ALTERNATING CURRENT | | | |
|---|---|---|---|---|
| **Substitute:** | Electricity | AC | DC | Energy |
| **Score:** | 0.5 | 1.0 | 0.0 | 0.3 |

## 4.2 Metrics

To evaluate a substitutability system, it is given each question from the dataset and made to calculate a substitutability score for each substitute-substitutee pair. We expect a system to give substitutability scores of real values between $0.0$ and $1.0$, $0.0$ being no substitutability and $1.0$ being perfect substitutability. Table 4.3 is an expected response from a system to the question in Table 4.2.

The tricky part now is to score/evaluate a system based on how *correct* or *incorrect* its response is when compared to the aggregated dataset. For this, we devised five metrics which give an agreement score between $0.0$ and $1.0$, $0.0$ being complete disagreement and $1.0$ being completely agreement with the dataset.

The following subsections contain many equations that formally define each metric. $D$ is used to represent the entire dataset of 88 questions and each $Q \in D$ is a particular question from the dataset. The coverage of each question is written as $cover(Q)$ and each individual substitute is denoted as $Q_i$ where $1 \leq i \leq 4$. A specific substitute's volunteer score is then written as $score(Q_i)$. When evaluating a substitutability system, each question is processed by that system to produce a substitutability score for each substitute. So in the context of any $Q$, $P = processed(Q)$, where $P$ is essentially a mirror of $Q$ but $score(P_i)$ gives the system's substitutability score instead of the volunteer score from the dataset.

Table 4.4: A question from the dataset without a clear winner.

| **Substitutee:** BRIGHT (coverage: 16) | | | | |
|---|---|---|---|---|
| **Substitute:** | Dull | Intelligent | Stupid | Smart |
| **Score:** | -13 | +11 | -14 | +14 |

## 4.2.1 Clear winner

The first metric tests whether a system can identify the best substitute when there is a clear winner in the question. Not all questions are suitable for this metric, as some have more than one good substitute. A question is considered to have a clear winner (an obvious best substitute) if the top volunteer score for any substitute is at least two-thirds of the question's coverage (number of total attempts) and all other runner-ups are scored below this threshold. Using the example in Table 4.2 again, `AC` is a clear winner as no other potential substitute has a volunteer score above 10; recall that the question has a coverage of 15.

In contrast, Table 4.4 is an example that does not have a clear winner. This question has a coverage of 16 and the best two substitutes are both scored above 10.

Equations (4.1) to (4.3) are used to determine clear winners in the dataset. $Best$ determines the highest volunteer score of any substitute, e.g. 14 for Table 4.4; $Top_v$ finds all substitutes that have a volunteer score above two-thirds of a question's coverage, `Smart` and `Intelligent` in this case; and $HasCW$ determines whether a question in the dataset has a single clear winner or not, which will be false for this example.

$$Best(Q) = \max_{i=1}^{4} score(Q_i) \tag{4.1}$$

$$Top_v(Q) = \{Q_i \big| score(Q_i) > \frac{2 \cdot cover(Q)}{3}, 1 \leq i \leq 4\} \tag{4.2}$$

$$HasCW(Q) = |Top_v(Q)| == 1 \tag{4.3}$$

After removing questions that did not have clear winners, we were left

Table 4.5: A set of system scores that disagrees with the dataset according to the clear winner metric.

| **Substitutee:** ALTERNATING CURRENT | | | | |
|---|---|---|---|---|
| **Substitute:** | Electricity | AC | DC | Energy |
| **Score:** | 0.7 | 0.9 | 0 | 0.3 |

with 62 questions. These questions were then evaluated against a substitutability system. For each question, the system either agrees (score of $1.0$) or disagrees (score of $0.0$) with the dataset. A system agrees if only one substitute is given a score that is above $\frac{2}{3}$ and that substitute is the same as the clear winner identified by the volunteers.

In Equation (4.4), $Top_s$ is similar to $Top_v$ but is specifically used for a system substitutability scores as opposed to volunteer scores. $CW(P, Q)$ in Equation (4.5) scores the actual agreement between a system against the dataset for a particular question $Q$ and its processed counterpart $P$.

$$Top_s(P) = \{P_i \big| score(P_i) > \frac{2}{3}, 1 \leq i \leq 4\} \tag{4.4}$$

$$CW(P, Q) = \begin{cases} 1, & \text{if } Top_s(P) == Top_v(Q) \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

The substitutability scores from Table 4.3 agrees with the dataset question – the runner-up substitute is scored at $0.5$ (below the threshold) and `AC` is correctly identified as the best substitute. Counter to this is a set of substitutability scores that would disagree with the dataset (Table 4.5, where `Electricity` has been given too high a substitutability score. Therefore its score sits above the threshold along with `AC`and there is no clear winner (i.e. $Top_s$ will return a set with more than one element).

An average is taken of all the agreements (as a set of ones and zeros) to give the final metric – average clear winners identified (**CW** in Equation (4.7)).

$$D_{CW} = \{Q \big| Q \in D, HasCW(Q)\} \tag{4.6}$$

$$CW = \frac{\sum_{Q \in D_{CW}} CW(P,Q)}{|D_{CW}|} \tag{4.7}$$

### 4.2.2 Good substitutes

The second metric tests whether a system can identify *good* substitutes, which includes clear winners as well as questions that contain multiple highly scored substitutes. In the dataset, a substitute is considered *good* if it has a volunteer score that is at least half of the question's coverage (number of volunteers that answered it). In other words, at least half of the volunteers agreed that the substitute is suitable. Using the "BRIGHT" example in Table 4.4, substitutes `Intelligent` and `Smart` are considered to be good substitutes, as they both have a volunteer score higher than eight (recall that the question had a coverage of 16).

In Equation (4.8), $Good_v$ determines all the good substitutes of a particular question in the dataset based on volunteer scores. $HasGS$ in Equation (4.9) determines whether a question has any good substitutes (i.e. at least one). Note that by this definition, all questions that have a clear winner qualifies for this metric.

$$Good_v(Q) = \{Q_i \big| score(Q_i) \geq \frac{cover(Q)}{2}, 1 \leq i \leq 4\} \tag{4.8}$$

$$HasGS(Q) = |Good_v(Q)| > 0 \tag{4.9}$$

Questions that do not have good substitutes are removed, which left 87 questions (only one question did not have good substitutes). For each of the 87 questions, the system being evaluated receives an agreement score between $0.0$ and $1.0$ that reflects the fraction of good substitutes identified. A substitute is identified as being good if the system gives it a substitutability score of at least $0.5$.

Table 4.6: A set of system scores that half agrees with the dataset according to the good substitutes metric.

| **Substitutee:** BRIGHT | | | | |
|---|---|---|---|---|
| **Substitute:** | Dull | Intelligent | Stupid | Smart |
| **Score:** | 0.0 | 0.4 | 0.0 | 0.7 |

In Equation (4.10), as with the previous metric, $Good_s$ is similar to $Good_v$ but is specifically used for a system's substitutability scores. $GS(P, Q)$ in Equation (4.11) scores the agreement between a system and the dataset for a particular question $Q$ and its processed counterpart $P$.

$$Good_s(P) = \{P_i \big| score(P_i) \geq 0.5, 1 \leq i \leq 4\} \tag{4.10}$$

$$GS(P, Q) = \frac{|Good_s(P) \cap Good_v(Q)|}{|Good_v(Q)|} \tag{4.11}$$

For example, consider a system which calculated the substitutability scores shown in Table 4.6 against the dataset question in Table 4.4. `Smart` has successfully been identified as a good substitute, but the system missed `Intelligent` (substitutability score is below $0.5$). Therefore it would receive an agreement of $0.5$, or $50\%$, for this question.

An average is taken of all the agreements to give the final metric – average good substitutes identified (**GS** in Equation (4.13)).

$$D_{GS} = \{Q \big| Q \in D, HasGS(Q)\} \tag{4.12}$$

$$GS = \frac{\sum_{Q \in D_{GS}} GS(P, Q)}{|D_{GS}|} \tag{4.13}$$

### 4.2.3 Bad substitutes

The third metric tests whether a system can successfully identify *bad* substitutes. That is, phrases which clearly cannot be used in place of the substitutee. A substitute is considered *bad* if it has a volunteer score that is

Table 4.7: A question from the dataset with bad substitutes.

| **Substitutee:** FAST (coverage: 12) | | | | |
|---|---|---|---|---|
| **Substitute:** | Quick | Slow | Big | Small |
| **Score:** | +12 | -9 | -10 | -10 |

sufficiently below zero. We decided that anything below a threshold of negative one-fifth of a question's cover is bad. This meant that more volunteers crossed-out the substitute (rejected it as being a suitable substitute) than those that circled it (believed it to be the best substitute).

In Equation (4.14), $Bad_v$ determines all the bad substitutes of a particular question in the dataset based on volunteer scores. $HasBS$ in Equation (4.15) determines whether a question has any bad substitutes (i.e. at least one).

$$Bad_v(Q) = \{Q_i \big| score(Q_i) < \frac{-cover(Q)}{5}, 1 \leq i \leq 4\} \tag{4.14}$$

$$HasBS(Q) = |Bad_v(Q)| > 0 \tag{4.15}$$

Table 4.7 is a question with a coverage of 12, so any substitute with a volunteer score less than -2 would be considered a bad substitute. I.e. `Slow`, `Big` and `Small`.

For these bad substitutes, we expect a substitutability system to produce a score that is zero or near-zero. We decided that any score below $0.1$ is acceptable. The agreement between a system and a particular question in the dataset is the fraction of bad substitutes it is able to identify, giving it an agreement score between $0.0$ and $1.0$.

In Equation (4.16), as with the previous metric, $Bad_s$ is similar to $Bad_v$ but is specifically used for a system's substitutability scores. $BS(P, Q)$ in Equation (4.17) scores the agreement between a system and the dataset, for a particular question $Q$ and its processed counterpart $P$.

Table 4.8: A set of system scores that two-thirds agrees with the dataset based on the bad substitutes metric.

| **Substitutee:** FAST | | | | |
|---|---|---|---|---|
| **Substitute:** | Quick | Slow | Big | Small |
| **Score:** | 0.85 | 0.0 | 0.15 | 0.03 |

$$Bad_s(P) = \{P_i \big| score(P_i) < 0.1, 1 \leq i \leq 4\} \tag{4.16}$$

$$BS(P,Q) = \frac{|Bad_s(P) \cap Bad_v(Q)|}{|Bad_v(Q)|} \tag{4.17}$$

Consider a system which produced the set of scores shown in Table 4.8 for the question in Table 4.7. `Slow` and `Small` have both been successfully identified as a bad substitutes, but the system missed `Big` (with a substitutability score above $0.1$). Therefore it would receive an agreement of $0.67$ (2dp), or two-thirds, for this particular question.

Questions that did not have bad substitutes, as previously defined, were removed, leaving 84 questions. Then each question is tested against a system to determine its average agreement towards the dataset for bad substitutes – average bad substitutes identified (**BS** in Equation (4.19)).

$$D_{BS} = \{Q \big| Q \in D, HasBS(Q)\} \tag{4.18}$$

$$BS = \frac{\sum_{Q \in D_{BS}} BS(P,Q)}{|D_{BS}|} \tag{4.19}$$

### 4.2.4 Combo

The previous two metrics, **GS** and **BS**, are great for penalising systems which fail to recognise suitable substitutes or incorrectly give unsuitable substitutes non-zero substitutability scores. However, on their own, each metric is flawed. If a system (`A`) gives a perfect score to every potential substitute (i.e. 1.0), then the **GS** metric will evaluate this system as having a perfect agreement to the dataset. Similarly, if a system (`B`) gives every substitute a score of 0.0, the **BS** metric would score this system as having perfect agreement with the dataset.

We decided that these two metrics needed to be combined into a single metric. While **GS** will evaluate system `A` with a perfect agreement of 1.0, it will fully penalise system `B` with an agreement of 0.0. Vice versa for **BS**. So combining the two metrics should give a fair indication towards a system's true performance.

One method is to simply combine the two metrics using their arithmetic mean (i.e. the average), but this type of mean is heavily influenced by high outliers. So if one of the metrics produces a trivially high agreement, the mean will drastically pulled up even if the opposing metric scores 0.0. An alternative is the harmonic mean [29], which is typically used when an average of rates is required (values between 0.0 and 1.0). It takes the reciprocal of the arithmetic mean of reciprocals as shown by Equation (3.1) back in Section 3.1.2. Harmonic mean tends strongly towards the smallest number, due to its reciprocal nature to the arithmetic mean. This is a highly desirable property, as it will punish systems that receive trivially high agreement scores in only one of the **GS** or **BS** metrics.

Equation (4.20) is a simplified version of harmonic mean when there are only two rates [21]. Combining rates in this manner is very similar to the F-measure [39], a frequently used metric in information retrieval and natural language processing that tests for general accuracy, where the two rates being combined are precision and recall. F-measure is also parameterised by a weight, which alters the final value to favour one rate over another. Our **Combo** metric most closely resembles the $F_1$-score, which is simply the traditional harmonic mean with even weighting.

$$Combo = 2 \cdot \frac{BS \cdot GS}{BS + GS} \tag{4.20}$$

We will use the example in Table 4.9 to demonstrate how the **Combo**

Table 4.9: A set of substitutability scores produced by three different systems.

| **Substitutee:** WATERMELON (coverage: 15) | | | | |
|---|---|---|---|---|
| **Substitute:** | Fruit | Melon | Honeydew | Rockmelon |
| **Volunteer Score:** | +4 | +11 | -10 | -11 |
| **Good/Bad:** | - | Good | Bad | Bad |
| **System X:** | 0.4 | 0.8 | 0.0 | 0.0 |
| **System Y:** | 0.3 | 0.7 | 0.2 | 0.0 |
| **System Z:** | 0.1 | 0.1 | 0.0 | 0.0 |

metric is used to evaluate a system against a particular question in the dataset. This question has a coverage of 15, which makes `Honeydew` and `Rockmelon` bad substitutes and `Melon` a good substitute (recall that a substitute is considered good if it has a volunteer score at least half of the question's coverage).

System X successfully identifies the good substitute and both bad substitutes, so it performances perfectly according to both metrics.

$$GS_X = 1.0$$
$$BS_X = 1.0$$
$$Combo_X = 1.0$$

System Y successfully identifies the good substitute, but over-values the bad substitute `Honeydew` (recall that a substitute is considered bad only if a system scores it below $0.1$). Therefore it is penalised by **BS**, which will also be reflected in the **Combo** metric.

$$GS_Y = 1.0$$
$$BS_Y = 0.5$$
$$Combo_Y = 0.67 \text{ (2dp)}$$

System Z successfully identifies both bad substitutes, but undervalues the good substitute, so it receives an agreement score of $0.0$ under the **GS** metric and is also penalised heavily by the **Combo** metric.

$$\begin{aligned} GS_Z &= 0.0 \\ BS_Z &= 1.0 \\ Combo_Z &= 0.0 \end{aligned}$$

We believe that **Combo** is a good overall metric for indicating a substitutability system's true performance, by punishing systems that receive trivially high agreements from either **GS** or **BS** on their own, whilst still providing a fair evaluation for systems that have a partial agreement to the dataset. Note that while the previous examples calculated **Combo** using a single question, the actual metric will use the average **GS** and **BS** agreements over the entire dataset.

### 4.2.5 Substitute ranking

The final metric measures how well a system ranks substitutes against each other in the context of a particular substitutee. More specifically, for every pair of substitutes in a question, can the system correctly identify which one is better? Consider the question in Table 4.10 from the dataset and a set of scores provided by two substitutability systems.

Table 4.10: A set of substitutability scores produced by two different systems.

| **Substitutee:** VIOLIN (coverage: 14) | | | | |
|---|---|---|---|---|
| **Substitute:** | Guitar | Instrument | Stringed Instrument | Orchestra |
| **Volunteer Score:** | -12 | -1 | +14 | -6 |
| **System X:** | 0.0 | 0.8 | 0.7 | 0.2 |
| **System Y:** | 0.9 | 0.5 | 0.8 | 0.1 |

The ranking of the substitutes is clear, as determined by the volunteer scores. `Stringed Instrument` is a better substitute than `Instrument` for the substitutee `Violin`, `Instrument` is better than `Orchestra` and `Orchestra` is better than `Guitar`.

A simple way to compare the ranking produced by a system against the volunteer's ranking is to check every pairwise comparison to see if the two agrees (there are six pairs in total, $\binom{4}{2} = 6$). Using this method, the two systems (`X` and `Y`) will achieve the following rank scores:

System X incorrectly values `Instrument` over `Stringed Instrument`, but gets everything else correct in terms of ranking order. Therefore, out of the six possible pairs of potential substitutes, System X only misranks one of them – receiving a substitute rank score of $0.83$ (2dp) or $\frac{5}{6}$. More formally, Equations (4.21) and (4.22) are used to make this calculation. Note that Equation (4.21) can be applied to both volunteer and system scores.

$$Comp(Q_i, Q_j) = \begin{cases} -1, & \text{if } score(Q_i) < score(Q_j) \\ 1, & \text{otherwise} \end{cases} \tag{4.21}$$

$$SR(P, Q) = \frac{\sum_{i=1,j=2,i<j}^{4} \begin{cases} 1, & \text{if } Comp(P_i, P_j) == Comp(Q_i, Q_j) \\ 0, & \text{otherwise} \end{cases}}{6} \tag{4.22}$$

System Y also makes one mistake, but this mistake has appointed the worst substitute `Guitar` to be the best substitute; a relatively large error when compared to System X. This is reflected by the pairwise comparisons, in which System Y fails half of them, receiving a substitute rank score of only $0.5$ (three out of six pairs correct).

Used in this manner, our metric is calculated in a similar fashion to the Kendall tau rank correlation coefficient [27] (based on the Kendall tau or bubble-sort distance) in the sense that it counts the number of concordant (agreement in rank order) and discordant (disagreement in rank order) pairs between two lists (see Equation (4.23)).

$$Kendall - tau = \frac{concordant - discordant}{no.of pairs} \tag{4.23}$$

However, being a ranking correlation coefficient, the Kendall's tau produces a value between $-1$ and $1$. A value of zero implies no correlation at all between two rankings ($0.5$ by our metric), while values below zero imply an opposing correlation. If a system produces an opposing ranking of substitute scores when compared to the dataset (i.e. a negative rank correlation), then inverting all of its substitutability scores ($inverted = 1 - score$) would effectively multiply its rank correlation by $-1$, thus improving it. Given this fact, it may appear strange that our substitute rank metric ranges from $0.0$ to $1.0$, considering values below $0.5$ are not helpful.

The reason behind our decision should become clear if we analyse the question and scores in Table 4.11. In this question, `Estimate` is clearly the best substitute for the substitutee `Approximate` according to the human volunteers, while the remaining three substitutes are all bad and ranked very closely together.

At first glance, System X seems to have done very well, it would certainly receive perfect agreement scores from all the previous metrics (**CW**, **GS**, **BS** and **Combo**). But on closer inspection, it has incorrectly ranked the three bad substitutes, however subtle the differences may be. So using our substitute rank metric in its current state would rate System X at just $0.5$ and the Kendall's tau coefficient would produce a value of $0.0$, or no correlation at all; a rather unfair result.

In contrast, System Y seems to do a very poor job on this question at a glance – it fails to identify the best substitute or the three bad substitutes (it would receive $0.0$ using all previous metrics) . But because it has correctly ranked the three bad substitutes in the correct order, which should be irrelevant, it will be rated as having performed better than System X. It correctly ranks four out of the six pairs, receiving a score of $0.67$ (2dp), or

Table 4.11: Substitutability scores that would be unfairly judged by a ranking correlation.

| **Substitutee:** APPROXIMATE (coverage: 16) | | | | |
|---|---|---|---|---|
| **Substitute:** | Reason | Estimate | Calculate | Process |
| **Volunteer Score:** | -6 | +16 | -5 | -7 |
| **System X:** | 0.07 | 0.9 | 0.0 | 0.1 |
| **System Y:** | 0.7 | 0.6 | 0.9 | 0.5 |

$0.33$ (2dp) using Kendall's tau coefficient ($\frac{4-2}{6}$).

To remedy this inherent unfairness, we decided that pairs of substitutes should be considered equally good (or bad) if their scores are similar. For the volunteer scores, substitutes that are scored with a difference no greater than one-fifth of the question's coverage are considered equal. For system scores, we used a threshold of 0.1. Note that volunteer scores can potentially range from $-cover(Q)$ to $+cover(Q)$, which is why we used a fifth of the coverage as the threshold instead of a tenth.

The equations from (4.24) to (4.26) have been adjusted to accommodate this new addition to the metric.

$$Comp_v(Q_i, Q_j) = \begin{cases} -1, & \text{if } score(Q_i) < score(Q_j) - \frac{cover(Q)}{5} \\ 1, & \text{if } score(Q_i) > score(Q_j) + \frac{cover(Q)}{5} \\ 0, & \text{otherwise} \end{cases} \tag{4.24}$$

$$Comp_s(P_i, P_j) = \begin{cases} -1, & \text{if } score(P_i) < score(P_j) - 0.1 \\ 1, & \text{if } score(P_i) > score(P_j) + 0.1 \\ 0, & \text{otherwise} \end{cases} \tag{4.25}$$

$$SR(P, Q) = \frac{\sum_{i=1,j=2,i<j}^{4} \begin{cases} 1, & \text{if } Comp_s(P_i, P_j) == Comp_v(Q_i, Q_j) \\ 0, & \text{otherwise} \end{cases}}{6} \tag{4.26}$$

These new set of equations form a metric that calculates a ranking agreement between the dataset and a system, with a small threshold for noise. Re-evaluating the example in Table 4.11, the question has a coverage of 16, so all three bad substitutes are considered equal (scores within three points of each other). System X also considers them equally bad, by scoring them within $0.1$ of each other, thus it receives a perfect agreement score of $1.0$.

System Y on the other hand, would completely disagree with the volunteer ranking under this new metric (i.e. all six pairs are in disagreement), receiving a punishing agreement score of $0.0$. This new metric also no longer behaves like a rank correlation. Even if we inverted the substitutability scores produced by System Y, it would barely increase its substitute rank agreement, as opposed to a perfect $1.0$ that is expected from a rank correlation.

Unlike the previous metrics, substitute ranking can be applied to all 88 questions. The final metric is then simply the average agreement score for each question – average substitute rank agreement (**SR** in Equation (4.27)).

$$SR = \frac{\sum_{Q \in D} SR(P, Q)}{|D|} \tag{4.27}$$

# Chapter 5

# Results and Analysis

In this chapter, we first analyse the agreement between the human volunteers who helped create our dataset. Then various existing systems are evaluated using the methods described in Section 4.2, along with our substitutability systems from Chapter 3. Next, we take a closer look at the major differences between WordSub and WikiSub. In the final part of this chapter, we focus on techniques that will allow our substitutability systems to be further optimised, both in terms of performance and speed.

All tests were run on a Windows machine with the specifications shown in Table 5.1 and all systems were written in Java. Any charts displayed in this chapter were created either using R [55] or Google Sheets.

Table 5.1: Test machine specifications.

| | |
|---|---|
| Processor | Intel Core i7-2600K @ 3.4GHz |
| Memory (RAM) | 8GB DDR3 |
| Operating System | Windows 7 Professional 64-bit |
| Java Runtime Environment | Version 6 Update 31 |

## 5.1 Human Agreement

Before we evaluate substitutability systems against our dataset, we wanted to acquire a benchmark of just how difficult the task is. To do so, we will calculate the human agreement of the volunteers against the dataset (i.e. how much the volunteers agreed with each other). This will provide an idea on how tricky the task is, even for humans, and also gives meaning to arbitrary agreement scores when we do come to evaluate substitutability systems.

By taking the raw answers from the volunteers, a benchmark agreement score can be calculated using each of the metrics from the previous section. However, recall that a volunteer's answers are excluded from the dataset if they do not recognise/understand one of the terms, so we will not evaluate the agreement of any answers where the volunteer has ticked the "Don't know all the terms" box (see Section 4.1). Substitutability systems will not have the same luxury. If a system cannot match a term to any targets, then it is forced to produce a conservative substitutability score of $0.0$, which will likely lower its agreement on that particular question. We reason that this is still fair, as the human agreement will be treated as an optimistic (best-case) agreement score – a ceiling performance which substitutability systems should aim for.

The volunteers answered each question by circling good (or the best) substitutes and crossing out bad ones. So functions such as $Top_s$ (4.4), $Good_s$ (4.10) and $Bad_s$ (4.16) can easily be adapted to measure human agreement. $Top_s$ and $Good_s$ will both return substitutes that were circled by the volunteer, while $Bad_s$ will return substitutes that were crossed out. Each metric can then be calculated as per the same process defined in Section 4.2.

Substitute ranking is the only exception to this procedure, as there is no explicit *ranking* in a volunteer's answers. Instead, we create an implied ranking by assigning real numbers to the answers. $0.8$ is given to any sub-

Table 5.2: Human agreement on dataset.

| **Metric** | **Agreement** |
|---|---|
| Clear Winner (CW) | 0.795 |
| Good Substitutes (GS) | 0.821 |
| Bad Substitutes (BS) | 0.600 |
| Combo | 0.693 |
| Substitute Ranking (SR) | 0.666 |

stitute that is circled, $0.0$ is given to any substitute that is crossed out, and $0.4$ is given to the others (unmarked substitutes). The selected values ensure that circled substitutes are above the *good* threshold of $0.5$, unmarked substitutes are below this threshold and crossed out substitutes fall within the *bad* threshold of $0.1$. A value of $0.8$ for circled substitutes also ensures that the **CW** metric functions correctly, requiring a score of at least $\frac{2}{3}$.

There is also an issue with the **BS** metric. On closer inspection of the raw answers, and from interviewing some of the volunteers, we discovered that many people were reluctant about crossing out a substitute unless they were certain of its unsuitability. Additionally, a small minority of volunteers did not cross out any substitutes at all across all the questions they answered. Substitutability systems behave in the exact opposite manner. Being conservative, they will label a substitute as *bad* when it is not certain of its suitability. A system will not have a clue regarding a substitute's unsuitability. To compensate for this bias, and to ensure that the human agreement is a truly optimistic score, we will exclude the evaluation of answers from volunteers who refused to cross out any substitutes. Note that this only affects the **BS** metric, the other metrics will still evaluate all volunteer answers.

Results from the evaluation of human agreement are listed in Table 5.2. As mentioned previously, these agreement scores are optimistic, which means any system that achieves similar agreement scores can be viewed as performing at least as well as an average human.

## 5.2 Performance Evaluation

### 5.2.1 Comparison with existing systems

We implemented five existing approximate string match systems in Java, which allowed us to evaluate our substitutability systems together with the existing systems on a common testing platform. Two of these systems are the current state-of-the-art approximate matching algorithms, both specifically designed for evaluating keyphrase extraction systems – **R-precision** and **Modified R-precision**. The other systems are the three newest techniques tested in [30] from the field of machine translation and summarisation – **BLEU**, **ROUGE** and **METEOR**. METEOR, similar to WordSub, also employs WordNet as a knowledge base. As such, we implemented the system with the help of the JWI library, in order to provide efficient access to WordNet's dictionary. We also used optimal values for METEOR's system parameters as presented in [35], tuned for English: $\alpha = 0.81$, $\beta = 0.83$ and $\gamma = 0.28$. See Chapter 2 for a detailed explanation on each of the existing systems.

Table 5.3 compares the performance of all the aforementioned systems using each of the metrics introduced in Section 4.2. It includes **GS** and **BS**, two metrics which are unreliable on their own as discussed previously, but are listed in the table to report the components of the **Combo** metric. Highlighted in bold, are the top agreement scores achieved by any system for the three relevant metrics – **CW**, **Combo** and **SR**.

It is immediately obvious that our two substitutability systems, WikiSub and WordSub, far outperform existing systems. This is not surprising when considering the composition of the dataset. Most of the questions require a system to make a semantic connection between the input terms in order to produce appropriate substitutability scores, but almost all of the existing systems rely on some form of lexical processing (i.e. string manipulation). So while they are able to suitably process input terms such as `applied science` $\rightarrow$ `computer science`, the same cannot be said

Table 5.3: Performance comparison of all systems.

| System | CW | Combo | SR | GS | BS |
|---|---|---|---|---|---|
| Human Agreement | 0.798 | 0.693 | 0.666 | 0.821 | 0.520 |
| WikiSub | 0.355 | **0.680** | **0.580** | 0.661 | 0.700 |
| WordSub | **0.356** | 0.609 | 0.508 | 0.483 | 0.823 |
| R-precision | 0.000 | 0.187 | 0.170 | 0.103 | 0.946 |
| Modified R-precision | 0.065 | 0.205 | 0.167 | 0.115 | 0.950 |
| BLEU | 0.000 | 0.000 | 0.152 | 0.000 | 0.980 |
| METEOR | 0.065 | 0.300 | 0.212 | 0.178 | 0.946 |
| ROUGE | 0.058 | 0.000 | 0.170 | 0.000 | 0.946 |

for input terms without any lexical overlap, such as `money` $\rightarrow$ `cash`. The majority of input terms in our dataset are of the latter category, which explains the relatively poor performance achieved by the existing systems. This is even more apparent when observing the extreme discrepancy between the **GS** and **BS** agreement scores. Failing to see any lexical overlap between input terms, the existing systems produce conservative scores of $0.0$, which results in a trivially high **BS** agreement. Simultaneously, the **GS** agreement is dragged downwards by the same notion.

METEOR deviates from the existing systems by exploiting WordNet synsets in a similar fashion to our system WordSub. As a result, METEOR is able to identify both lexical and semantic overlaps between input terms. Thus it outperforms other existing system under the **Combo** and **SR** metrics, while matching Modified R-precision under the **CW** metric. But this is limited to direct synonyms that reside within WordNet synsets. Without any exploration beyond synsets, METEOR, while outperforming other existing systems, still struggles with our dataset.

However, the higher performance obtained by our substitutability system does come at the cost of speed. WordSub runs on the order of 100x slower than the lexically-oriented systems (e.g. R-precision, BLEU etc.) and 10x slower than METEOR. WikiSub on the other hand, with its significantly larger knowledge base, has to spend much longer processing each input – on the order of 800x slower than WordSub, at 8.5 seconds per input. Note that these processing times are from versions of our substitutability systems that have already implement all the speed optimisations discussed in Section 5.3. Please refer to that section for exact processing times. We also noted that most volunteers completed their sheet of 10 questions within five minutes. Since each question containing four substitute-substitutee pairs, WikiSub processes at a similar speed to humans. But being an automatic evaluator, the time, energy and monetary costs associated with organising domain experts for manual evaluation is avoided.

In the real life context of evaluating keyphrase extraction systems, being able to perform timely evaluations is important. We can estimate how long this process will take by using the SemEval-2010 workshop [31] as an example. 19 substitutability systems participated in the workshop, each were trained and evaluated against a testing set of 100 documents. The entire evaluation process required the comparison of a vast number of input pairs, on the order of half a million. WordSub, with an average processing time of approximately one-hundredths of a second per input, would take just over an hour to evaluate all the systems; a perfectly acceptable time frame. However, WikiSub would require over a month of continuous processing for the same task, which is simply infeasible.

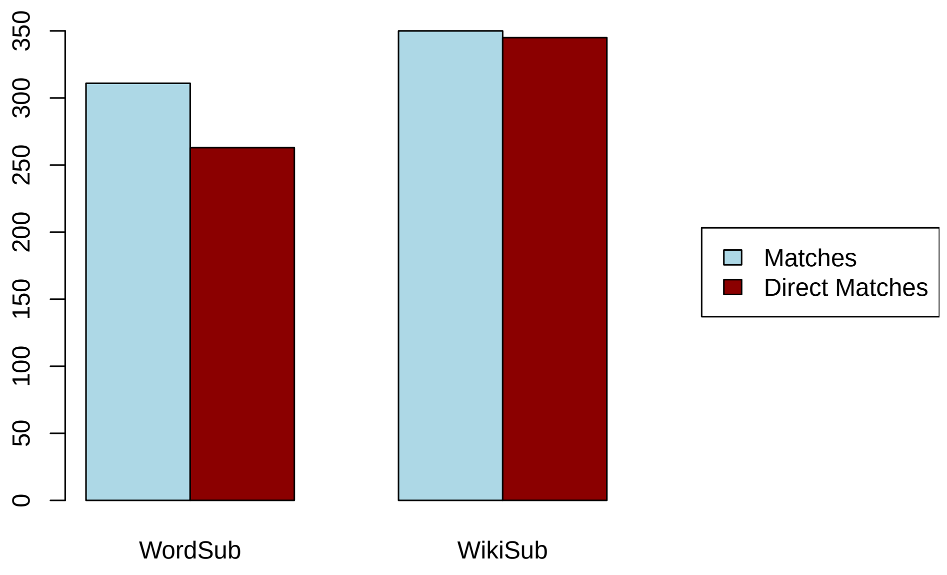
The top of Table 5.3 also lists the optimistic human agreement scores, which all sit comfortably above the best system agreements. Under the **CW** metric, the best system result (achieved by WordSub) sits at an unimpressive $44.5\%$ of the human agreement. We believe this may have been a consequence of how we specified the metric, leading to a high human agreement. Recall that a question has a clear winner when there is only a single candidate substitute that is scored above two-thirds of the question's coverage. So by definition, such questions require at least two-thirds of the volunteers to agree on what the best substitute is, thus arriving at an inherently high human agreement. Under the two other relevant metrics, the best system results sit much closer to the human agreement. At $98.1\%$ of the human agreement under the **Combo** metric and $87.1\%$ under the **SR** metric; both achieved by WikiSub.

### 5.2.2 WordSub vs. WikiSub

When implementing the two substitutability systems, we believed that a larger, more diverse knowledge base would provide better performance. Two reasons were hypothesised for this. First, a larger knowledge base contains a greater number of entities, which should result in higher quality targets that match directly to the input terms. A smaller knowledge base may have to frequently break a term into aspects, or fail to match any targets all together. Second, a larger knowledge base is likely to offer more potential transitions between entities, allowing the system to find substitution paths that would have otherwise been missing in a smaller knowledge base.

Figure 5.1 provides clear proof that our first hypothesis is correct. Using Wikipedia, the larger knowledge base compared to WordNet, WikiSub is able to match significantly more input pairs to targets than WordSub. Out of the 352 total pairs in our dataset, WikiSub matches 350 of them ($99.4\%$), while WordSub matches only 311 ($88.4\%$). This means WordSub has to give a conservative substitutability score of $0.0$ for $11.6\%$ of inputs from the dataset. Moreover, WikiSub can *directly* match 345 input pairs to targets ($98.0\%$), while WordSub is only able to directly match a mere 263 pairs ($74.7\%$). So for one in every four input pairs from the dataset, WordSub has to either resort to breaking a term up into its aspects or giving up and producing a trivial substitutability score of $0.0$. In contrast, WikiSub only has this problem for one in every 50 input pairs. Section 5.3.6 further explains the downside of having to match targets to aspects instead of directly to the full term itself.

Figure 5.1: Number of input pairs successfully matched to targets.



When a system, such as WordSub, resorts to scoring input pairs conservatively, it fails to identify substitutes which are in fact *good*. This flaw is reflected accordingly by a decrease in the **GS** agreement score. Seen in Table 5.3, WikiSub significantly outperforms WordSub by almost 37% under this metric – $0.661$ vs $0.483$.

Following the same rationale, WikiSub should similarly outperform WordSub under the **CW** metric. But this is not the case. WikiSub receives a **CW** agreement score that is in fact slightly worse than WordSub, albeit the difference is a negligible $0.001$. So why does WikiSub not have the superior performance that we expect under this metric? After all, the **CW** metric is essentially the same as the **GS** metric, but evaluated on a smaller subset of questions (see Section 4.2.2). We turned to our second hypothesis in the hopes of finding an explanation.

In terms of options for transitions, Wikipedia certainly offers plenty more than WordNet. On average, entities from Wikipedia exhibit five to 10 times the number of transitions than WordNet entities. For example, the category `Cameras` boasts nine sub-categories, 99 child articles and two parent categories, giving a total of 110 potential transitions. WordNet on the other hand, only has nine synsets leading out of the synset `[camera, photographic camera]` via hyponym pointers and one other synset can be reached via a hypernym pointer, giving a total of only 10 potential transitions.

The **CW** metric is actually implicitly two-sided, in a similar fashion to the **Combo** metric. But this two-sidedness is less formally defined than the individual components of **Combo** – **GS** and **BS**. On one side, the system must first correctly ascertain that the clear winner substitute, as identified by the human volunteers, does in fact have high substitutability (i.e. there exists a relatively short substitution path to it). This task is helped greatly by the increased number of entities and transitions present in Wikipedia, as reflected by the higher **GS** agreement. The other side of **CW** requires the system to register the other three substitutes in the question as being

Table 5.4: A question which punishes noisy transitions.

| **Substitutee:** PUBLIC TOILET (coverage: 17) | | | | |
|---|---|---|---|---|
| **Substitute:** | Toilet | Room | Powder Room | Bedroom |
| **Volunteer Score:** | +16 | -3 | -3 | -13 |
| **WordSub:** | 0.70 | 0.27 | 0.11 | 0.00 |
| **WikiSub:** | 1.00 | 0.68 | 0.00 | 0.00 |

subpar, with substitutability scores that fall below the threshold of $\frac{2}{3}$. It is this second side of **CW** which penalises systems such as WikiSub.

Even though WordNet offers less transitions, the ones it does have tend to stay focused towards the emitting entity. Transitions are strict and well-defined, presenting little chance for the search process to drift unexpectedly in meaning. However, as the number of available transitions increase, so too does the likelihood of encountering *noisy* transitions. Admittedly, a greater number transitions will naturally introduce more potentially appropriate entities for the search process to explore, but it equally introduces potentially inappropriate entities as well. The existence of this *noise* within WikiSub's knowledge base causes it to sporadically over-value certain input pairs that should otherwise have a lower substitutability score according to human judgement.

Consider the question in Table 5.4 and the substitutability scores calculated by WordSub and WikiSub. Both systems correctly score the best substitute `Toilet` above the clear winner threshold of $\frac{2}{3}$, but WikiSub also over-values the runner up substitute `Room`, putting it above the clear winner threshold along with `Toilet`. Such a mistake would not have affected the agreement of **GS**, but is punished severely by **CW** with a score of $0.0$.

A closer inspection of the substitution paths found by the respective systems explains why this occurs. Due to the highly-connected structure of Wikipedia, WikiSub's path passes from the Wikipedia category `Rooms` directly to its sub-category `Public toilets` via a single Specialisation transition. On the other hand, with its more strict, less-connected structure, WordNet requires two Specialisation transitions for the same input, moving through the entity `Toilet` along the way. The result is an appropriately lower substitution score than the score calculated by WikiSub.

Having more available transitions can negatively impact **CW** in another way also. Recall that the transition score of a Specialisation is heavily influenced by its breadth (the number of other possible Specialisation transitions from the more generic entity), as defined in Section 3.1.1. Therefore, the abundance of transitions offered by Wikipedia will typically result in a lower than expected substitutability score. In limited cases, the subsequent score of the best substitute will be lowered enough to fall below the clear winner threshold. The bottom line is, unwanted noise is introduced by the plethora of transitions available in Wikipedia. Consequently, WikiSub is liable to manufacture substitutability scores which deviate more from the expected value than a system which uses a smaller, stricter knowledge base.

Despite its few shortcomings, WikiSub still performs markedly better than WordSub, with its superior access to a larger number of entities and transitions. However, the comparatively expensive processing time, makes it difficult to justify this increase in performance. In Section 5.3.6, we propose a system which could combine the best traits of both of our substitutability systems – taking WordSub's speed along with WikiSub's performance.

## 5.3 Improvements and optimisations

This section covers a range of optional enhancements to the general design of a substitutability system (as presented in Section 3.1), both in terms of performance/agreement and speed. Some of these enhancements have already been added to one or both of our systems, WordSub and WikiSub, while others are left for future work.

### 5.3.1 Truncated search

The two most costly operations (in terms of time) of a substitutability system are FINDTARGETS in Algorithm 1 and SEARCH in Algorithm 4. All other operations are negligible in comparison. We wanted to get a sense of exactly how much time the system is spending on each operation. So we used WordSub as a reference point, by tracking its processing times when being evaluated against the dataset. Recall that the dataset consists 88 questions, each containing four substitute-substitutee pairs, resulting in a total of 352 individual runs of the substitutability system during each complete pass of the dataset.

After 10 complete passes of the dataset, we found that WordSub spent an average time of $317.7$ milliseconds to process each substitute-substitutee pair. However, only an average of $0.9$ milliseconds was spent on the FIND-TARGETS procedure, a mere $0.3\%$ of the total. Therefore, it makes sense to first optimise the SEARCH procedure.

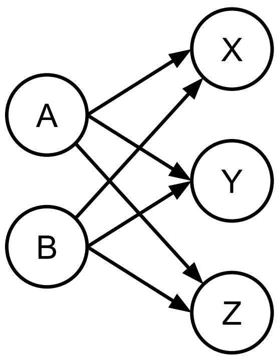
On closer inspection, we realised that WordSub struggles with input pairs that have low or no substitutability. Such inputs require the system to perform a near-exhaustive search of the entire knowledge base (WordNet), just in case there exists some long, obscure path linking the inputs. More often than not, these pairs are not substitutable at all, or have so little substitutability that it is not worth scoring it. If WordSub can recognise this particular scenario, it should truncate the search early to avoid exploring fruitless paths. To achieve this, we introduce a threshold score, whereby a node will no longer be expanded (have its neighbours explored further) if it has a score lower than the threshold – the **stopping threshold**. This change is added to the if-statement in line 16 of Algorithm 4, replacing $0.0$ with the stopping threshold.

The stopping threshold needs to be a non-zero value that is high enough to reduce the average search time, but also low enough to not affect the overall performance of the system (i.e. WordSub should receive the same agreement from the evaluation metrics). After trying out a range of values, we found that the performance was not affected as long as the stopping threshold is below $0.1$. Staying on the side of caution, we opted for a value of $0.08$.

Using this slight modification to the SEARCH procedure, we completed another 10 passes of the dataset using WordSub. This time, the total average time spent processing each input pair was only $33.9$ milliseconds, a reduction of almost 90%. FINDTARGETS now took up $2.7\%$ of the overall processing time (still at $0.9$ milliseconds), which is still relatively small. So we continued to explore ways of reducing the cost of SEARCH.

### 5.3.2 Multi-search

Figure 5.2: Searches required for multi-aspect inputs.



Recall that certain terms generate a list of target sets, as opposed to a single target set. This occurs when the term itself cannot be matched directly to an entity in the knowledge base, and instead has to be broken down into aspects as per Algorithm 2. Input pairs that contain these *difficult* terms will require multiple search passes, one for each unique pair of aspects. For example, consider an input pair where the substitute has to be broken down into two aspects and the substitutee into three aspects (see Figure 5.2). A total of six searches would be required, but many of the searches share the same starting point, either from aspect `A` or aspect `B`.

When more than one search originates from the same point, it is a waste to restart the entire search from scratch each time. So WordSub was modified to handle searches from a single origin target set to multiple destination target sets, all in a single search pass which we called MULTISEARCH. Algorithm 5 outlines this new procedure.

To get a baseline, we timed the *single* SEARCH version of WordSub against a subset of the dataset that only contained *difficult* input pairs (containing terms that had to be broken down into multiple aspects). There are 48 of such pairs out of 352 in the dataset. On average, WordSub spent 130.3 milliseconds in the SEARCH procedure for these difficult pairs. This is almost four times over the average for the whole dataset, which was to be expected. After adjusting WordSub to use MULTISEARCH, the average time decreased to 90.9 milliseconds, a noticeable reduction of 30%. Retesting against the entire dataset, the average processing time per input pair has decreased to 27.3 milliseconds.

### 5.3.3 Caching

The next step is to see if we can further optimise MULTISEARCH. We profiled our Java implementation using a tool called VisualVM [50] and found that a significant portion of time was spent on the EXPAND subroutine (line 23 in Algorithm 5). This was not surprising as EXPAND directly queries the knowledge base. In the case of WordSub, EXPAND uses JWI (see Section 3.2) to access WordNet pointers/relations. During a full pass over the dataset, WordSub revisits many of the synsets from previous searches. This means that the same synset is expanded multiple times, each requiring costly calls to the knowledge base, which is a waste of time. So to speed things up, we altered WordSub to maintain a cache of all the pointers that have been expanded during searches, as well as the transition scores associated with each expansion. EXPAND is then able to check this cache first and directly return the desired result if it already exists,

**Algorithm 5** Finds the best paths of substitutability between from the target set $ts1$ to a list of target sets $tss2$. Returns the score of each path.

```
    procedure MultiSearch($ts1, tss2$)
        $fringe \leftarrow$ max heap, $visited \leftarrow$ empty set, $scores \leftarrow [\ ]$
        fill $scores$ with -1s to the same length as $tss2$
        populate $fringe$ with $ts1$ converted to nodes
5:      while $fringe$ is not empty do
            $node \leftarrow$ poll from $fringe$
            if $node.entity$ already in $visited$ then
                continue
            else
10:             for $i \leftarrow 0, tss2.length - 1$ do
                    if $scores[i] == -1$ and $tss2[i]$ contains $node$ then
                        $scores[i] \leftarrow node.score$                ▷ found a path
                    end if
                end for
15:             if no -1s left in $score$ then                    ▷ all paths found
                    break
                end if
            end if
            add $node.entity$ to $visited$
20:         if $neigh.score < 0.08$ then                      ▷ stopping threshold
                continue
            end if
            $transitions \leftarrow$ Expand($node.entity$)
            for all $tr$ in $transitions$ do
25:             if $tr.next$ not in $visited$ then       ▷ the entity reached by $tr$
                    $neigh \leftarrow$ ToNode($tr.next$)
                    $neigh.score \leftarrow node.score * tr.score$
                    offer $neigh$ to $fringe$
                end if
30:         end for
        end while
        replace any -1s with 0s in $scores$
        return $scores$
    end procedure
```

without having to make expensive queries to the knowledge base.

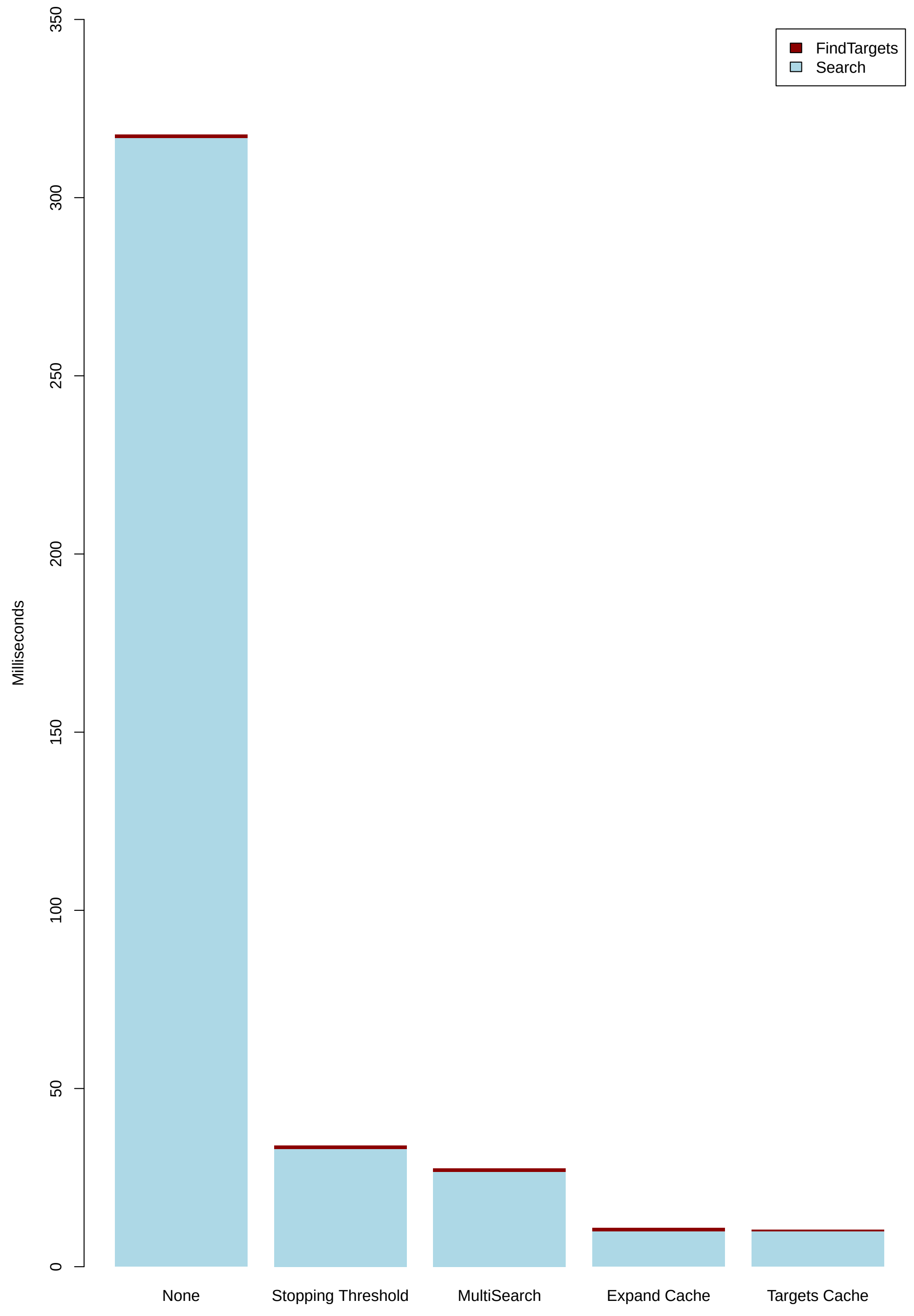
When using substitutability systems such as WordSub in a real life context, entities will also be frequently revisited. Therefore, this particular optimisation is not specific to just being able to process a dataset of input pairs quickly. For example, WordSub may be used to evaluate the overall substitutability of a list of candidate keyphrases against a gold standard, in order to test the performance of a keyphrase extraction system. In doing so, each candidate keyphrase has to be tested against each gold standard keyphrase. Furthermore, all the keyphrases will likely be closely related to each other, as they all came from the same document. As a result, the chance of revisiting entities will be high, and so the hit-rate of the cache will be high also.

Timing our system (with caching) once again against the dataset, WordSub managed to reduce its average processing time per input pair to just $10.9$ milliseconds. This is yet another significant reduction of a further 60% (down from $27.3$ milliseconds).

With the search process optimised to the best of our abilities, we turned our heads to the FINDTARGETS procedure. Still sitting at an average of $0.9$ milliseconds per input, it now took up $8.3\%$ of the overall processing time, high enough to warrant a closer look. Once again, we profiled this procedure using VisualVM and found that most of the time was spent inside the sub-routine MATCHTARGETS. Just as with EXPAND, MATCHTARGETS makes costly queries to the knowledge base in its attempts to match a plain text phrase to entity targets. This is especially true for multi-aspect inputs, where MATCHTARGETS has to be called repeatedly to test out all potential aspect split points.

Since caching worked so well when optimising the sub-routine EXPAND, we tried it again with MATCHTARGETS. WordSub now maintains a second cache of all target sets returned from MATCHTARGETS, allowing the sub-routine to return immediately when it receives a hit from the cache. With this final alteration in place, we timed the system once more

Figure 5.3: Average processing time per input for WordSub optimisations.



using our dataset. The results were clear – for the FINDTARGETS subroutine, WordSub now only spends $0.4$ milliseconds on average per input, a reduction of over half, down from $0.9$ milliseconds. The overall average processing is now $10.4$ milliseconds per input pair. This is minuscule when compared to the original $317.7$. By implementing all the speed optimisations discussed in this section, WordSub reduced its overall processing time by $96.7\%$. Figure 5.3 is a stacked bar plot summarising the time savings achieved at each stage of optimising WordSub.

### 5.3.4 Speed optimisations on WikiSub

WikiSub has a much long processing time than WordSub, because it uses a larger knowledge base. WordNet dwarfs in size when compared to Wikipedia. Without any speed optimisations, WikiSub takes on average two to four minutes to process non-trivial input terms (inputs which result in substitution paths of length greater than one). In the case where a near-exhaustive search of Wikipedia is performed (input terms that have little or no substitutability), WikiSub often fails to terminate as the fringe and visited sets of the search grow beyond the memory capacity of our test machine (see Table 5.1). Therefore, speed optimisations were necessary just to make the system functional.

Modifying WikiSub's SEARCH procedure to use a stopping threshold (introduced in Section 5.3.1), the average processing time was reduced to an acceptable 15 seconds per input. It also guaranteed that the system could always run to completion well before it ran into memory issues. Next, we altered WikiSub to use the MULTISEARCH procedure in the same manner described in Section 5.3.2. Doing so further reduced the average processing time to $12.3$ seconds per input.

The final speed optimisation to be added is caching, as discussed in Section 5.3.3. However, this proved to be tricky. With Wikipedia being so vast, maintaining caches once again resulted in memory issues. So Wiki-

Sub had to frequently remove the least used entries from the cache whenever it came close to running out of memory. Consequently, the speed increase from the caches was not as impressive as when applied to WordSub. The final average processing time, after implementing all speed optimisation on WikiSub, is $8.5$ seconds per input pair, still almost a factor of 27 slower than the unoptimised version of WordSub.

### 5.3.5 Optimum transitions

The final substitutability score given to any input pair is calculated as the product of all the transition scores along the best substitution path (see Section 3.1.4). Conversely, the best substitution path is defined to be the path which maximises substitutability (has the highest product of transition scores). This means that the transition scores themselves influence the resulting path found by the system, which directly affects the final substitutability score.

In the initial design of our systems, we chose transitions scores based on common sense intuitions and a bit of trial-and-error. But now that we have a dataset specifically designed for evaluating substitutability, it is possible to alter the current transition scores to optimise a system's performance on the dataset. Once again, we decided to experiment on WordSub, since it is much faster to work with. Table 5.5 lists the set of transition scores used by WordSub, which will be optimised in order to improve the system's overall performance on the dataset. These can also be thought of as adjustable parameters to the system.

Manually adjusting each transition score proved to be unrealistic. Instead, we employed a simulated annealing algorithm [32] to perform this task automatically. Simulated annealing is a generic algorithm used for approximating the global optimum of a large search space, where an exhaustive search is too costly. It is an adaptation of the Metropolis-Hastings algorithm [43], originally proposed by Nicholas Metropolis in 1953. Metropolis-

Table 5.5: Initial transition scores used by WordSub.

| **Transition** | **Score** |
|---|---|
| $Sim$ – Similar/See Also | 0.5 |
| $Der$ – Derivation | 0.7 |
| $Sp_{min}$ – Specialisation Minimum | 0.2 |
| $Sp_{max}$ – Specialisation Maximum | 0.99 |
| $Gn_{max}$ – Generalisation Maximum | 0.99 |

Hastings (MH) is a Markov chain Monte Carlo (MCMC) [56] technique to sample from a distribution for which direct sampling is difficult (e.g. the unknown distribution formed by five adjustable transition scores used by WordSub).

Simulated annealing differs from MH in that it is designed for hill climbing, not sampling. The algorithm begins by picking a random starting point $s$ in the search space. In WordSub's case, this is simply a set of random transition scores (between $0.0$ and $1.0$) with the exception that $Sp_{min} \leq Sp_{max}$. Then for each iteration, $s$ is moved in a random direction by a random amount, to the point $s'$. We implemented this step by moving one of the five transitions scores (chosen at random) by a value drawn from a normal distribution [52] (denoted as $x \sim \mathcal{N}(\mu, \sigma^2)$) and then clamping the new value within valid ranges as shown in Equations (5.1).

$$\begin{aligned} NewScore &= \max(0, \min(1, OldScore + x \sim \mathcal{N}(0, 0.25))) \\ NewSp_{min} &= \max(0, \min(Sp_{max}, OldSp_{min} + x \sim \mathcal{N}(0, 0.25))) \\ NewSp_{max} &= \max(Sp_{min}, \min(1, OldSp_{max} + x \sim \mathcal{N}(0, 0.25))) \end{aligned} \tag{5.1}$$

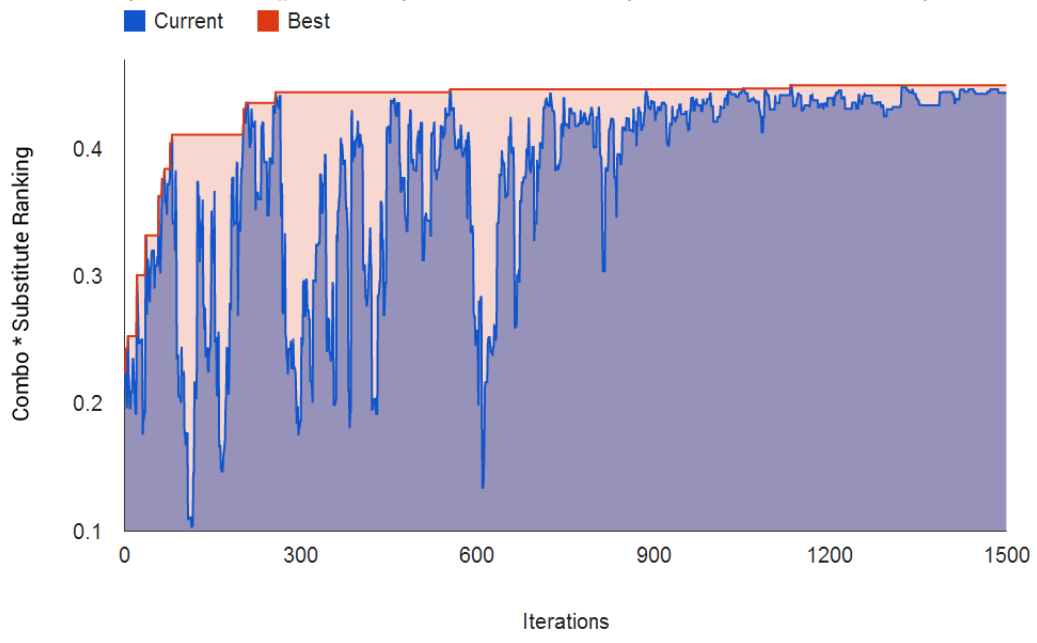
Next, the algorithm decides to either move to the new point $s'$ or stick with the current point $s$. The probability of accepting the new point is calculated using Equation (5.2), where the function $metric$ produces an agreement score based on one or a combination of the metrics introduced in Section 4.2. $T$ is the *temperature*, a key concept in the thermodynamic analogy used by simulated annealing. When the temperature is non-zero, there is a chance of accepting a new point even if it is worse than the current one. This lowers the chance of the algorithm getting stuck at local optima. As the temperature decreases, it becomes less likely that the algorithm will accept a new point that performs worse than the current one. In fact, with a temperature of zero, simulated annealing is effectively the same as greedy hill climbing. Note that because transition scores influence the way a node expands to its neighbours, the cache added to WordSub in Section 5.3.3 had to be cleared at the start of each iteration.

$$acceptance = \mathrm{e}^{\frac{metric(s') - metric(s)}{T}} \tag{5.2}$$

The algorithm is repeated for a set number of iterations, reducing the temperature each time until it reaches zero or near-zero. We began with a temperature of $2.0$ and reduced it by a factor of $0.995$ each iteration for a total of 1500 iterations, resulting in a final temperature of $0.001$ (3dp). The set of transition scores that corresponded to the best performing iteration is then selected as an approximation of the global optimum. For the $metric$ function, we used the product of the **Combo** and **SR** metrics, as we wanted to optimise the transition scores in relation to both (we observed that using just one of the metrics had a detrimental effect on the performance under the other metric). However, the metrics were not used against the entire dataset. Out of the 88 questions, 20 contained at least one term that WordSub could not match to any target entities (even after breaking it down into aspects). When encountering such terms, WordSub defaults to a conservative substitutability score of $0.0$, which is a result that isn't affected by transition scores. Optimising/training against such questions will not improve performance. Therefore, we decided to exclude these 20 questions (leaving 68 questions for *training*) when optimising WordSub's transition scores, for the same reason human volunteers were asked to only attempt questions where they understood all the terms (see Section 4.1).

Despite starting with a temperature of $2.0$, we were unconvinced that

Figure 5.4: Optimising WordSub using simulated annealing.



a single simulated annealing run would produce a set of transition scores sufficiently close to the global optimum. So we restarted the algorithm with a new random starting point at the end of each run. 50 full passes of simulated annealing was conducted over a period of two weeks, totalling 75,000 iterations and over 60 hours of processing time. Figure 5.4 shows the result of a single run of 1500 iterations. As expected, the performance of each iteration fluctuates wildly at the start, when the temperature is high and the Markov Chain explores all parts of the search space even if performance drops. But as the temperature nears zero, the Markov Chain settles around an optimum solution. In Figure 5.4's case, the best solution actually occurred at the 1133rd iteration, after which the search got trapped at a local optimum due to the near-zero temperatures.

Taking the best set of transition scores obtained during the 50 simulated annealing passes, we re-evaluated WordSub against the entire dataset using this new set of transition scores. The optimised scores are displayed in Table 5.6 under the column labelled **Score***. Most of the optimised tran-

Table 5.6: Optimised transition scores for WordSub.

| Transition | Score | Score* |
|---|---|---|
| $Sim$ – Similar/See Also | 0.5 | 0.534 |
| $Der$ – Derivation | 0.7 | 0.644 |
| $Sp_{min}$ – Specialisation Minimum | 0.2 | 0.675 |
| $Sp_{max}$ – Specialisation Maximum | 0.99 | 0.930 |
| $Gn_{max}$ – Generalisation Maximum | 0.99 | 0.915 |

Table 5.7: WordSub performance after optimising transition scores.

| Metric | WordSub | WordSub* | Improvement |
|---|---|---|---|
| Clear Winner (CW) | 0.356 | 0.419 | +17.7% |
| Good Substitutes (GS) | 0.483 | 0.592 | +22.6% |
| Bad Substitutes (BS) | 0.823 | 0.790 | -4.0% |
| Combo | 0.609 | 0.677 | +11.2% |
| Substitute Ranking (SR) | 0.508 | 0.542 | +6.7% |

sition scores fall relatively close (within $0.1$) to our initial estimates, with the exception of $Sp_{min}$, which differs significantly by $0.475$. We believe this is due to the fine-grained nature of WordNet's hierarchical structure, where the information loss is lessened when transitioning towards specificity. Table 5.7 summarises the performance improvement over the original system (**WordSub**) when using optimised transition scores (**WordSub***). There is a notable improvement in performance under almost every metric. The exception being **BS**, which has a small decrease in performance of $4.0\%$. However, this is more than offset by the significant improvement of $22.6\%$ under the **GS** metric. The outcome is a respectable **Combo** agreement of $0.677$, which comes very close to matching WikiSub's agreement of $0.680$. Similar conclusions can be drawn by observing the **SR** agreement, coming behind WikiSub by only $0.038$, while further increasing its lead over WikiSub under the **CW** metric.

With such positive results, we conclude that, given a suitable training set, simulated annealing is an effective technique for optimising the transition scores of a specific knowledge base. It has enabled a system, which is backed by a relatively small knowledge base (i.e. WordSub), to become competitive against a system which is backed by a much larger knowledge base (i.e. WikiSub). Note that the results displayed here only demonstrate the potential for improvement. Since the transitions scores were optimised on a subset of the dataset used to also evaluate it, we cannot conclusively compare it to other systems. The disadvantage of using this method, like many other MCMC algorithms, is that the process takes a long time. We were able to obtain good results within an acceptable time frame only after including all the time saving improvements discussed in the previous sections for WordSub. Running simulated annealing on a system backed by a much larger knowledge base such as WikiSub, despite the same speed optimisations, would prove to be unrealistic. A rough calculation puts the approximate processing time of $50 \times 1500$ iterations on the order of 6 years.

### 5.3.6 Multiple knowledge bases

Using optimum transition scores whilst searching for substitution paths can result in a notable improvement in dataset agreement, but it relies on the correct matching of input terms to target sets. The procedure responsible for this task is FINDTARGETS in Algorithm 1. If line 13 is triggered, then the input term was matched directly to a set of targets. This means that the knowledge base contains an entity that perfectly reflects the input term, so the match is of a high quality. On the other hand, if line 11 is triggered, then the input term failed to match any entity and the system is forced to return a conservative score of $0.0$. Line 8 falls somewhere in between, when the input term has to be broken down in to two or more aspects before a match could be found. While this is much better than no match at all, the quality of the match may be questionable. For example, consider the phrase `acts of god` and assume a direct match cannot be

Table 5.8: WordSub performance on different subsets of questions.

| Metric | Direct only | Non-empty | All |
|---|---|---|---|
| Clear Winner (CW) | 0.543 | 0.469 | 0.419 |
| Good Substitutes (GS) | 0.786 | 0.709 | 0.592 |
| Bad Substitutes (BS) | 0.711 | 0.740 | 0.790 |
| Combo | 0.746 | 0.724 | 0.677 |
| Substitute Ranking (SR) | 0.703 | 0.625 | 0.542 |

found. If it were to be broken down into the aspects `[acts][god]`, then some of the original meaning of the term would have been lost.

To test out this theory, we evaluated WordSub (using optimum transition scores from Section 5.3.5) against three subsets of the dataset. The first is a subset of 50 questions, where all the terms could be matched directly to targets by WordSub. The second is a subset of 68 questions, where all the terms could either be matched directly to targets, or had to be split into aspects. The last subset is simply the entire dataset of 88 questions. Table 5.8 lists the results of WordSub's performance for the three subsets in respective order. As expected, the performance drops with each successive subset of questions under all metrics. With the exception of **BS**, but this is no surprise as it is susceptible to giving trivially high agreement scores to systems that give conservatively low substitutable scores (as discussed in Section 4.2.3).

Whether a term can be matched directly to targets is dependent on the knowledge base used by the system. One knowledge base is likely to contain entities that are missing from another knowledge base, and vice versa. For example, consider the input pair `consensus reality` $\rightarrow$ `mother wit`. `Consensus reality` can be matched directly to a target by WikiSub, returning a Wikipedia article with the same name. But WikiSub cannot do the same with `mother wit`, unless the term is split into aspects, which is something we would like to avoid. Conversely, WordSub cannot match `consensus reality` to targets without breaking it down into aspects, but it has no problem with `mother wit`, being an existing word in WordNet. What this shows is that knowledge bases can complement each other. While an individual system using a single knowledge base might struggle with particular input terms, using two or more knowledge bases in conjunction could lead to superior results.

We propose a system that is able to search for substitution paths across multiple knowledge bases. For the previous example, it will find a path from a Wikipedia target set to a WordNet target set. At some point during this search, the system has to move from a Wikipedia entity to a WordNet entity. For this, we introduce a new type of transition – an **Inter-knowledge** transition. To implement this fifth transition, the system will convert an entity back into its plain text form, then feed this into a MATCHTARGETS procedure which is attached to a different knowledge base. The returned results will then allow the system to transit the search to the new knowledge base. Note that by using MATCHTARGETS, we avoid aspects all together when making such transitions; only direct matches are allowed. We also reason that the quality of Inter-knowledge transitions falls somewhere between a Same transition to a Similar transition (a transition score of $0.5 - 1.0$). The specific number would depend on the strictness of MATCHTARGETS.

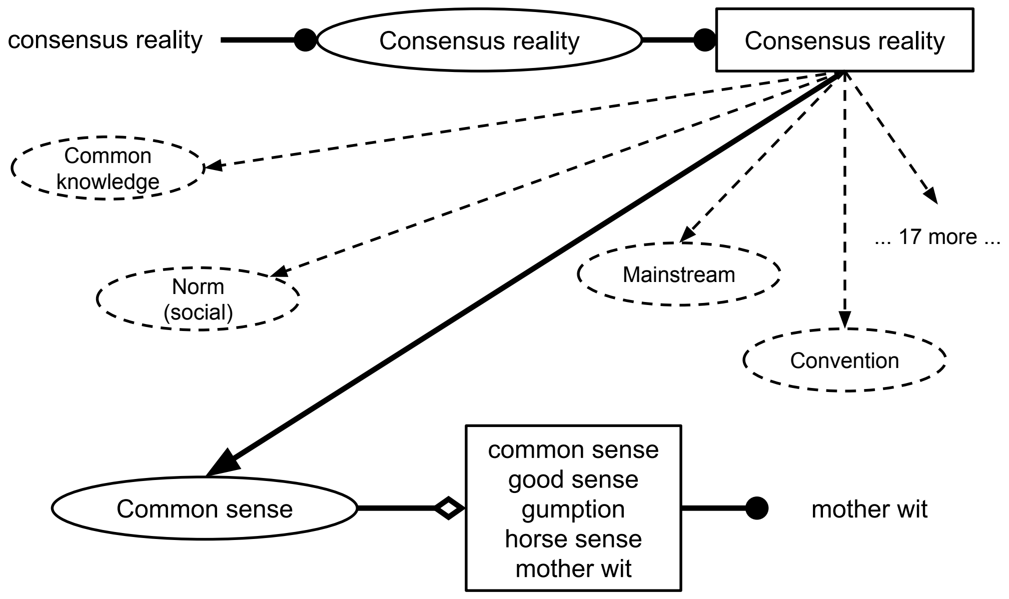
Going back to the previous example, this new system will first begin the search from the Wikipedia article `Consensus reality`. Moving through the hierarchical structure of Wikipedia, the search will soon reach the article titled `Common sense`. At this point, it will explore the option of crossing knowledge bases by feeding the phrase `common sense` into the MATCHTARGETS procedure for WordNet. Doing so will produce a synset which contains both `common sense` and `mother wit`, thus successfully completing the search, without the need of splitting any term into aspects.

Figure 5.5 displays the final substitution path, where the diamond-headed arrow represents the Inter-knowledge transition. Using the original transition scores set out in Section 3.1.3 and assuming a transition score of $0.9$ for the Inter-knowledge transition, the substitutability score of this path can be calculated as follows:

$$\begin{aligned}
\texttt{Consensus reality} \rightarrow \texttt{C:Consensus reality} &= 1.0\\
\texttt{C:Consensus reality} \rightarrow \texttt{Common sense} &= 0.48\\
\texttt{Common sense} \rightarrow \texttt{[common sense, ..., mother wit]} &= 0.9\\
\texttt{consensus reality} \rightarrow \texttt{mother wit} &= \mathbf{0.43}
\end{aligned}$$

Inter-knowledge transitions can also have asymmetric transition scores, which will weigh the search process to favour one knowledge base over another. This is especially useful when one knowledge base is significantly faster than the other and speed is of concern. For example, WordSub has a vastly superior processing time when compared to WikiSub (see Section 5.2), but lacks the extensive collection of entities and transitions that WikiSub has access to. So in order to exploit the best of both systems, an Inter-knowledge transition may encourage a move towards WordNet over Wikipedia, but fall back to following Wikipedia transitions when the search gets stuck on WordNet's side. More specifically, a transition from Wikipedia to WordNet will have a higher transition score than the opposing transition from WordNet to Wikipedia. This transition will be addressed in our future work (see Section 6.2).

Figure 5.5: A substitution path containing an Inter-knowledge transition.



# Chapter 6

# Conclusions

In this chapter, we conclude the work presented in this thesis. First, the major contributions made by this research are summarised. Then, further areas of exploration are proposed as future work, along with recommendations of how each one could be approached.

## 6.1 Contributions

The motivation of this thesis is to devise a method of automatically evaluating automatic keyphrase extraction systems that can mimic the quality of human judgements as if the evaluation was performed manually. We pinpointed the problem to the matching criteria used when determining the true positive rate of a keyphrase extraction system. Conventionally, the true positive rate is calculated by identifying exact, or near-exact string matches between candidates (produced by the system) and expected gold standards (produced by humans). However, this type of evaluation takes a pessimistic view of the system, and only measures the lower bound of a system's true performance. In order to mimic human judgements, the actual meaning contained in strings (semantics) must be considered.

To include the appropriate semantics into the string matching criteria, we introduced the notion of substitutability as a subclass of semantic relatedness that focuses on a smaller subset of relationships. Our first major contribution lies in the formal definition and quantification of substitutability, defining it to be *a context-free measure of how much information is retained when one phrase is used in place of another, whilst making as few assumptions as possible.* Based on this definition, we then designed a generalised system that can compute the substitutability of any two input terms, by exploiting the interlinking structure of external knowledge sources.

This system performs two main steps to accomplish the task. First, the input terms, which are expected to be in plain text form, are matched to entities in a knowledge base. An entity is defined to be a basic unit of knowledge within the knowledge base, and the entities which match to the initial inputs are defined as target sets. Second, the system attempts to find a path of maximal substitutability between the two target sets, moving from the substitute (first input term) to the substitutee (second input term). This step is achieved through a best-first search on the implicit graph structure inherent in the knowledge base. As the search expands from one entity to another, information retention from the original term is reduced. We take this into account by assigning transition scores to the movement from one entity to the next. These scores are real numbers between $0.0$ and $1.0$ depending on the type of transition, where $1.0$ implies perfect information retention and $0.0$ implies total information loss. The final substitutability score between two terms is then calculated as the product of all transition scores along the path of maximal substitutability. We categorised transitions into the following four types:

**Same** transitions have near-perfect information retention, such as terms that have the same stem or are closely-related synonyms.

**Similar** transitions have good information retention, where the two entities are related but do not have the same meaning.

**Specialisation** transitions are a movement towards specificity. The exact information retention depends on how common the substitution is, which is computed based on the breadth of the transition (the number of other possible specialisations).

**Generalisation** transitions are a movement away from specificity. The information retention is calculated in a similar manner to Specialisation transitions, but the scores are much smaller. Because Generalisation transitions make inherent assumptions, so we penalise them heavier when compared to Specialisation transitions.

Based on this generalised design of a substitutability system, we developed two concrete implementations of this design, which forms our second major contribution. The first system, WordSub, employed WordNet as its knowledge base. Entities were represented by synsets (sets of tightly-coupling synonyms that share a common concept) and were matched to the inputs using JWI, the MIT Java WordNet Interface. Transitions were represented by WordNet pointers, that form relationships between words and synsets. **Same** transitions were implicitly included by using synsets instead of individual words as the system's entities. **Similar** transitions were implemented by grammatical derivation, "see also" and "similar to" pointers. **Specialisation** transitions were implemented by hyponym type and hyponym instance pointers, while **Generalisation** transitions followed hypernym pointers. The second system, WikiSub, employed Wikipedia as its knowledge base. Entities were represented by articles (distinct concepts in human knowledge) and categories (groups of similar articles or concepts), which were matched to inputs using Wikipedia Miner, a Java toolkit released by the University of Waikato. **Same** transitions were implemented following redirects between articles and moving between a category and its central article. **Specialisation** transitions were implemented by exploring the child articles and sub-categories of a category, while **Generalisation** transitions were implemented by inspecting parent categories.

Our third major contribution involves the evaluation of approximate string matching systems, including both lexically and semantically oriented systems. We created a dataset which isolated the task of measuring substitutability, with the help of 130 volunteers, which consisted of university students ranging from undergraduate to doctorate. The dataset contained 88 questions, each of which presented four substitutes against a single substitutee, resulting in a total of 352 substitute-substitutee pairs that can be used for testing. For each question, volunteers were asked to identify the best substitute (out of the four available) to be used in place of the substitutee, or best substitutes in the case where it was hard to decide a single best substitute. They were also asked to identify any substitutes that were definitely unsuitable. After two months worth of data collection and entry, we aggregated the answers from the volunteers to assign a score that ranks the four substitutes against each other in light of a particular question's substitutee. In order to evaluate the performance of approximate string matching systems, we presented each system with the same task as the volunteers and measured its agreement to human judgements using the following metrics:

**Clear Winner (CW)** – When the human judgement indicates that one substitute is significantly better than the other three for a specific question, how often does the system also come to the same conclusion?

**Good Substitute (GS)** – How often the system agree with human judgements on which substitutes are *good*?

**Bad Substitute (BS)** – How often the system agree with human judgements on which substitutes are *bad*?

**Combo** – A combination of **GS** and **BS** by taking their harmonic mean, thus catching out systems which receive trivially high agreement scores for only one of the two metrics.

**Substitute Ranking (SR)** – The degree of agreement between the system and human judgements on the overall ranking of the four substitutes in each question.

Using these metrics, we evaluated the performance of our substitutability systems against existing approximate string matching techniques – R-precision, Modified R-precision, BLEU, METEOR and ROUGE. Our systems came out on top by a clear margin, especially against the lexical-based approaches that relied on string manipulation alone. Such non-semantic approaches struggled greatly with our dataset, as it was created with semantics in mind. METEOR, a system that did consider semantics in its matching criteria, performed notably better than the others, but our systems were still comfortably ahead of it. We also calculated the agreement between the volunteers who helped create the dataset, which was used as an optimistic upper-bound agreement score. Comparing this score against our two substitutability systems, we found that our systems' agreement was sensibly close to the optimistic human agreement. Therefore, we conclude that by leveraging external knowledge sources, automatic evaluation systems can have comparable quality to that of human judgements made in manual evaluation. As a result, such evaluation systems can produce consistently repeatable results, whilst avoiding the cost and energy of organising human evaluators.

## 6.2 Future work

There are many ways in which the contributions made in this thesis can be further extended. This section will cover three areas that we believe are worth exploring further.

The first of these is to implement and test further substitutability systems that use other knowledge bases. A promising source is Wiktionary [48], a side project started by the WikiMedia Foundation [25] near the end of 2002. Like WordNet, Wiktionary maintains a synonymy network of terms (words and common phrases), but unlike WordNet, it is multilingual, spanning over a hundred different languages. Another benefit of

Wiktionary lies in its crowd-sourced nature, which allows it to evolve over time, following the natural progression of human knowledge.

Wiktionary also provides an inbuilt method for implementing the Inter-knowledge transition proposed in Section 5.3.6. Instead of converting entities to plain text and re-invoking the MATCHTARGETS sub-routine, most Wiktionary entries link directly to their equivalent Wikipedia articles. Furthermore, the same Wikipedia articles also link back to Wiktionary. Thus, Inter-knowledge transitions can be implemented by simply following these explicit links for moving reliably between Wikipedia and Wiktionary.

In terms of speed, we reason that Wiktionary will have a comparable processing speed to WordNet, as they are similar in both structure and size. Therefore, as discussed in Section 5.3.6, the search routine in such a hybrid system should favour transitions within or into Wiktionary (the fast knowledge base) over Wikipedia. A depth-limited approach could be taken to achieve this effect. For example, an initial search is attempted using only Wiktionary. If no suitable substitutability paths are found, then the search is repeated with the inclusion of Inter-knowledge transitions. Since the processing time associated with Wikipedia dwarfs that of other knowledge bases (see Section 5.2.1), any overlap in the second search pass should be negligible.

A second direction of furthering this thesis lies in the exploration of alternative methods for evaluating approximate string matching systems. The format of our current dataset (questions containing four substitutes each against a single substitutee) is only one of many possible ways for isolating the task of computing substitutability. No doubt other formats could prove to be just as effective or potentially be even better. One approach we considered involves making volunteers compare two sets of substitute-substitutee pairs against each other, to decide whether one substitution is better than the other or are both similar in quality (i.e. $\texttt{A} \rightarrow \texttt{B}$ vs. $\texttt{C} \rightarrow \texttt{D}$).

This comparative approach once again avoids the need for humans to explicitly assign scores to the individual substitution pairs, which as we mentioned in Section 4.1, is not as reliable or consistent. Once the process of manual labelling is complete, a dataset can be created by constructing an ordered list of substitution pairs based on the data gathered from the human volunteers. Approximate string matching systems can then be evaluated using a ranking agreement, similar in fashion to our **SR** metric (see Section 4.2.5).

Evaluation can also be taken a step further, to go beyond substitutability as an isolated task. Systems could be tested in the context of a real-life IR evaluation task, such as evaluating the quality of automatically extracted keyphrases. More specifically, approximate string matching systems should be integrated as the matching criteria of the IR evaluation process described in Chapters 1 and 2. This less strict matching criteria will cause an increase in the true positive rate when compared to exact string matching, and subsequently result in higher overall performance scores (i.e. precision, recall and F-measure) given to the IR systems being evaluated. However, currently, there is no good way to reliably measure the *correctness* of these elevated scores.

One solution is to have humans judge the quality of sets of candidates against sets of gold standards, just as one would undertake in a manual evaluation process. For the specific example of keyphrase extraction, humans would label/score the quality of a set of candidate keyphrases (generated by real-life keyphrase extraction systems) against a set of professionally or author assigned keyphrases (e.g. from manually-indexed document collections such as FAO [11] and SemEval-2010 [31]). These human quality judgements can then be used to compare against the performance scores produced by automatic evaluation systems with integrated approximate string matching, as a kind of *meta-evaluation*.

The third area of future work concerns the optimisation of transition scores as explained in Section 5.3.5. These scores directly affect the final substitutability score given to any pair of input terms, thus having a significant impact on the overall agreement of the system when compared to human judgements. Transition scores can be *learnt* by tuning them against a dataset to maximise the performance over a certain metric. We used simulated annealing [32] to accomplish this task, but the process can be time-intensive and often got stuck at local optima, despite the measures taken to avoid it. Part of the problem could be due to the fact that we optimised our system over the product of two metrics (**Combo** and **SR**), forcing it to compromise between the two potentially conflicting measures. A dataset that relies on only a single metric, such as the rank agreement based one mentioned previously, could produce more consistent results.

Other hill climbing techniques should also be tried and tested, as they may prove to be more effective. Some promising methods include Particle Swarm Optimisation (PSO [28]) and Genetic Algorithm (GA [23]). Both PSO and GA maintain a population of candidate solutions, each requiring an evaluation pass over the dataset to calculate its fitness, before proceeding to the next iteration. To speed up this evaluation process, we propose that the entire population be evaluated in a parallel manner over the dataset. In other words, every candidate solution is first evaluated against the same question in the dataset, and their fitness/agreement for that question is recorded individually. The process is then repeated for every other question until the entire dataset has been covered. We believe this will be faster than a serial approach, as it should substantially increase the chance of a cache hit when utilising the caching technique introduced in Section 5.3.3. However, one slight change needs to be made to the caching implementation. The cache we implemented recorded the transition scores associated with each expansion, but because each candidate solution contains its own unique set of transition scores, the scores must be recalculated each time.

# Bibliography

[1] BARKER, K., AND CORNACCHIA, N. Using noun phrase heads to extract document keyphrases. In *Advances in Artificial Intelligence*. Springer, 2000, pp. 40–52.

[2] BARRIÈRE, C., AND JARMASZ, M. Keyphrase extraction: enhancing lists.

[3] CILIBRASI, R., AND VITÁNYI, P. M. Clustering by compression. *Information Theory, IEEE Transactions on 51*, 4 (2005), 1523–1545.

[4] CILIBRASI, R. L., AND VITANYI, P. M. The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on 19*, 3 (2007), 370–383.

[5] CLEARY, J. G., AND WITTEN, I. Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on 32*, 4 (1984), 396–402.

[6] CRABTREE, D. W., ANDREAE, P., AND GAO, X. Exploiting underrepresented query aspects for automatic query expansion. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007), ACM, pp. 191–200.

[7] DEUTSCH, L. P. Deflate compressed data format specification version 1.3.

[8] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[9] DUMAIS, S. T. Latent semantic analysis. *Annual review of information science and technology 38*, 1 (2004), 188–230.

[10] EL-BELTAGY, S. R. Kp-miner: A simple system for effective keyphrase extraction. In *Innovations in Information Technology, 2006* (2006), IEEE, pp. 1–5.

[11] FAO, J., AND FOODS, M. H. I. Food and agriculture organization of the united nations. *Rome, URL: http://faostat. fao. org* (2004).

[12] FELLBAUM, C. Co-occurrence and antonymy. *International journal of lexicography 8*, 4 (1995), 281–303.

[13] FINKELSTEIN, L., GABRILOVICH, E., MATIAS, Y., RIVLIN, E., SOLAN, Z., WOLFMAN, G., AND RUPPIN, E. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web* (2001), ACM, pp. 406–414.

[14] FINLAYSON, M. A. Code for java libraries for accessing the princeton wordnet: Comparison and evaluation.

[15] FRAKES, W. B. Term conflation for information retrieval. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval* (1984), British Computer Society, pp. 383–389.

[16] GABRILOVICH, E., AND MARKOVITCH, S. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI* (2007), vol. 7, pp. 1606–1611.

[17] GAILLY, J.-L., AND ADLER, M. The gzip compressor, 1991.

[18] GOLUB, G., AND KAHAN, W. Calculating the singular values and pseudo-inverse of a matrix. *Milestones in Matrix Computation: The*

*selected works of Gene H. Golub with commentaries: The selected works of Gene H. Golub with commentaries* (2007), 237.

[19] HARISPE, S., RANWEZ, S., JANAQI, S., AND MONTMAIN, J. Semantic measures for the comparison of units of language, concepts or entities from text and knowledge base analysis. *arXiv preprint arXiv:1310.1285* (2013).

[20] HASSAN, S., AND MIHALCEA, R. Semantic relatedness using salient semantic analysis. In *AAAI* (2011).

[21] HAVIL, J., AND GAMMA, J. Exploring euler's constant, 2003.

[22] HIRST, G., AND ST-ONGE, D. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database 305* (1998), 305–332.

[23] HOLLAND, J. H. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[24] HULTH, A. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing* (2003), Association for Computational Linguistics, pp. 216–223.

[25] INC., W. F. Wikimedia downloads, July 2011.

[26] JOTY, S., CARENINI, G., AND NG, R. Automatic topic labeling in asynchronous conversations.

[27] KENDALL, M. G. A new measure of rank correlation. *Biometrika* (1938).

[28] KENNEDY, J., EBERHART, R., ET AL. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks* (1995), vol. 4, Perth, Australia, pp. 1942–1948.

[29] KENNEY, J., AND KEEPING, E. Harmonic mean. *Mathematics of Statistics, Pt. 1, 3rd ed., Van Nostrand, Princeton, NJ* (1962), 57–58.

[30] KIM, S. N., BALDWIN, T., AND KAN, M.-Y. Evaluating n-gram based evaluation metrics for automatic keyphrase extraction. In *Proceedings of the 23rd international conference on computational linguistics* (2010), Association for Computational Linguistics, pp. 572–580.

[31] KIM, S. N., MEDELYAN, O., KAN, M.-Y., AND BALDWIN, T. Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation* (2010), Association for Computational Linguistics, pp. 21–26.

[32] KIRKPATRICK, S., VECCHI, M., ET AL. Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680.

[33] KOLMOGOROV, A. N. On tables of random numbers. *Theoretical Computer Science 207*, 2 (1998), 387–395.

[34] LAVIE, A., AND AGARWAL, A. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation* (2007), Association for Computational Linguistics, pp. 228–231.

[35] LAVIE, A., AND AGARWAL, A. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation* (2007), Association for Computational Linguistics, pp. 228–231.

[36] LEACOCK, C., AND CHODOROW, M. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database 49*, 2 (1998), 265–283.

[37] LI, M., CHEN, X., LI, X., MA, B., AND VITÁNYI, P. M. The similarity metric. *Information Theory, IEEE Transactions on 50*, 12 (2004), 3250–3264.

[38] LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop* (2004), pp. 74–81.

[39] MAKHOUL, J., KUBALA, F., SCHWARTZ, R., AND WEISCHEDEL, R. Performance measures for information extraction. In *In Proceedings of DARPA Broadcast News Workshop* (1999), pp. 249–252.

[40] MARTIN, A., AND PRZYBOCKI, M. The nist 1999 speaker recognition evaluationan overview. *Digital signal processing 10*, 1 (2000), 1–18.

[41] MEDELYAN, O. *Human-competitive automatic topic indexing*. PhD thesis, The University of Waikato, 2009.

[42] MEDELYAN, O., AND WITTEN, I. H. Thesaurus based automatic keyphrase indexing. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries* (2006), ACM, pp. 296–297.

[43] METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., AND TELLER, E. Equation of state calculations by fast computing machines. *The journal of chemical physics 21*, 6 (1953), 1087–1092.

[44] MIHALCEA, T., AND TARAU, P. T. Bringing order into texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (2004).

[45] MILLER, G. A. Wordnet: a lexical database for english. *Communications of the ACM 38*, 11 (1995), 39–41.

[46] MILNE, D., AND WITTEN, I. H. An open-source toolkit for mining wikipedia. *Artificial Intelligence 194* (2013), 222–239.

[47] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of machine learning*. MIT Press, 2012.

[48] NAVARRO, E., SAJOUS, F., GAUME, B., PRÉVOT, L., SHUKAI, H., TZU-YI, K., MAGISTRY, P., AND CHU-REN, H. Wiktionary and nlp: Improving synonymy networks. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources* (2009), Association for Computational Linguistics, pp. 19–27.

[49] OLSON, M. A., BOSTIC, K., AND SELTZER, M. I. Berkeley db. In *USENIX Annual Technical Conference, FREENIX Track* (1999), pp. 183–191.

[50] ORACLE. Java VisualVM (version 1.3.7). `http://visualvm.java.net/`, Jan. 2014.

[51] PAPINENI, K., ROUKOS, S., WARD, T., AND ZHU, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), Association for Computational Linguistics, pp. 311–318.

[52] PATEL, J. K., AND READ, C. B. *Handbook of the normal distribution*, vol. 150. CRC Press, 1996.

[53] PEDERSEN, T., PATWARDHAN, S., AND MICHELIZZI, J. Wordnet::similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004* (2004), Association for Computational Linguistics, pp. 38–41.

[54] PORTER, M. F. An algorithm for suffix stripping. *Program: electronic library and information systems 14*, 3 (1980), 130–137.

[55] R CORE TEAM. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[56] ROBERT, C., CASELLA, G., ET AL. A short history of markov chain monte carlo: subjective recollections from incomplete data. *Statistical Science 26*, 1 (2011), 102–115.

[57] SALTON, G., WONG, A., AND YANG, C. S. A vector space model for automatic indexing. *Commun. ACM 18*, 11 (Nov. 1975), 613–620.

[58] SINGHAL, A. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull. 24*, 4 (2001), 35–43.

[59] STORER, J. A., AND SZYMANSKI, T. G. Data compression via textual substitution. *Journal of the ACM (JACM) 29*, 4 (1982), 928–951.

[60] STRUBE, M., AND PONZETTO, S. P. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI* (2006), vol. 6, pp. 1419–1424.

[61] TERRA, E., AND CLARKE, C. L. Frequency estimates for statistical word similarity measures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1* (2003), Association for Computational Linguistics, pp. 165–172.

[62] TURNEY, P. Mining the web for synonyms: Pmi-ir versus lsa on toefl.

[63] TURNEY, P. Coherent keyphrase extraction via web mining.

[64] VAN RIJSBERGEN, C. Information retrieval. dept. of computer science, university of glasgow. *URL: citeseer. ist. psu. edu/vanrijsbergen79information. html* (1979).

[65] VAN RIJSBERGEN, C. Information retrieval: theory and practice. In *Proceedings of the Joint IBM/University of Newcastle upon Tyne Seminar on Data Base Systems* (1979), pp. 1–14.

[66] WAN, X., AND XIAO, J. Collabrank: towards a collaborative approach to single-document keyphrase extraction. In *Proceedings of*

*the 22nd International Conference on Computational Linguistics-Volume 1* (2008), Association for Computational Linguistics, pp. 969–976.

[67] WANG, D. X., GAO, X., AND ANDREAE, P. Dikea: domain-independent keyphrase extraction algorithm. In *AI 2012: Advances in Artificial Intelligence*. Springer, 2012, pp. 719–730.

[68] WELCH, T. A. A technique for high-performance data compression. *Computer 17*, 6 (1984), 8–19.

[69] WEST, D. B., ET AL. *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[70] WITTEN, I., AND MILNE, D. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA* (2008), pp. 25–30.

[71] WITTEN, I. H., PAYNTER, G. W., FRANK, E., GUTWIN, C., AND NEVILL-MANNING, C. G. Kea: Practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries* (1999), ACM, pp. 254–255.

[72] WU, Z., AND PALMER, M. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics* (1994), Association for Computational Linguistics, pp. 133–138.

[73] ZESCH, T., AND GUREVYCH, I. Approximate matching for evaluating keyphrase extraction. In *Proceedings of the 7th International Conference on Recent Advances in Natural Language Processing* (2009), Citeseer, pp. 484–489.