UNIVERSITY OF
CAMBRIDGE

Proposition-based summarization with a coherence-driven incremental model

Yimai Fang

Hughes Hall

This dissertation is submitted in January 2018
for the degree of Doctor of Philosophy

Abstract

Summarization models which operate on meaning representations of documents have been neglected in the past, although they are a very promising and interesting class of methods for summarization and text understanding. In this thesis, I present one such summarizer, which uses the proposition as its meaning representation.

My summarizer is an implementation of Kintsch and van Dijk's model of comprehension, which uses a tree of propositions to represent the working memory. The input document is processed incrementally in iterations. In each iteration, new propositions are connected to the tree under the principle of local coherence, and then a forgetting mechanism is applied so that only a few important propositions are retained in the tree for the next iteration. A summary can be generated using the propositions which are frequently retained.

Originally, this model was only played through by hand by its inventors using human-created propositions. In this work, I turned it into a fully automatic model using current NLP technologies. First, I create propositions by obtaining and then transforming a syntactic parse. Second, I have devised algorithms to numerically evaluate alternative ways of adding a new proposition, as well as to predict necessary changes in the tree. Third, I compared different methods of modelling local coherence, including coreference resolution, distributional similarity, and lexical chains.

In the first group of experiments, my summarizer realizes summary propositions by sentence extraction. These experiments show that my summarizer outperforms several state-of-the-art summarizers. The second group of experiments concerns abstractive generation from propositions, which is a collaborative project. I have investigated the option of compressing extracted sentences, but generation from propositions has been shown to provide better information packaging.

Contents

Chapter 1

Introduction

Summarization is one of the ultimate goals in artificial intelligence (AI) that will capture people's attention until it is solved. It is not merely a practical application of natural language processing (NLP), but is in fact entwined with many fundamental questions of the field, such as the representation of meaning and knowledge.

Spärck Jones (1999) devises a model about how an intelligent, deep summarizer would digest text into a meaning representation, manipulate that meaning representation, and generate new short text from the manipulated meaning representation (Figure 1.1). But how are we to arrive at the meaning representation that will be used to perform the operations of text understanding? It is a difficult question that has occupied researchers for many years, and different solutions have been found.

One of the most attractive meaning representations for this purpose is *proposition*, a small meaning unit that encapsulates roughly one predicate or one modification. The meaning expressed by a proposition can be interpreted relatively

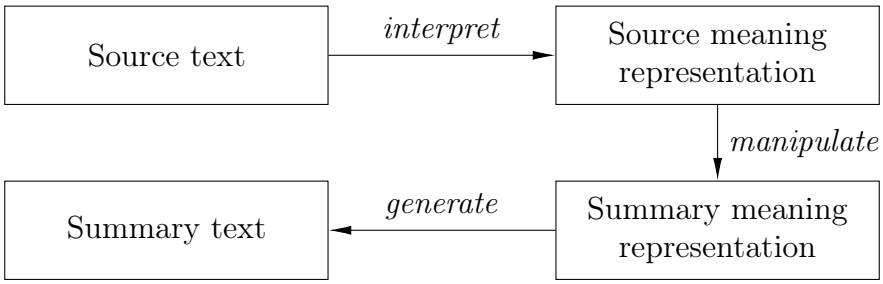


Figure 1.1: The process of summarization based on meaning representation

independently, and can be realized in language as a short sentence.[1] This property makes the proposition a type of content units suitable for summarization, which in its simplest form concerns the selection of content units.

To illustrate the idea of propositions, let us consider this sentence in a paragraph about artificial islands:

> *The technology of constructing artificial islands has been advanced considerably by the construction of Kansai Airport, which cost $20 billion and took seven years to complete.*

The information conveyed by the sentence can be represented by four propositions, which I roughly denote as:

1. OF (TECHNOLOGY, CONSTRUCT (ARTIFICIAL ISLAND))
2. ADVANCE (BUILD (KANSAI AIRPORT), TECHNOLOGY)
3. COST (BUILD (KANSAI AIRPORT), $20 BILLION)
4. DURATION (BUILD (KANSAI AIRPORT), 7 YEARS)

In the process of creating a summary, many types of manipulations are available. The simplest of these would be to drop propositions. If we follow this route, then there is a good reason why we should drop propositions 3 and 4. The focus of the overall text is about artificial islands, therefore proposition 1 and 2 provide closer connections to the context than the other two propositions. This idea is called *local coherence*, the property of pieces of text to build on each other. If a text is coherent, it has an internal logic and reads fluently.

Kintsch and van Dijk (1978, henthforth KvD) propose a model of text comprehension which manipulates propositions according to discourse coherence. In this model, the content of a document is processed incrementally, sentence by sentence. Local coherence is realized as the principle of *argument overlap*, i.e. new propositions are connected to old propositions with which they share arguments (components of a proposition). After all propositions are processed, a summary can be formed based on the best connected propositions.

The text pieces were selected with the coherence of the source text in mind, and they should, if all goes well, also result in new summary text also obeying

[1] Even if some of the sentences thus created would appear as unnatural, because they are too short.

local coherence constraints. This thesis investigates to which degree this promise holds for real world texts.

Following the KvD model, I aim to create an explanatory summarization system. Explainable AI has become increasingly important as applications of AI are integrated into daily life. It is sought after by many research initiatives in the world, e.g. the DARPA Explainable AI Initiative,[2] and is generally expected by the society, which is reflected, for example, in the General Data Protection Regulation (GDPR) legislation. Furthermore, explainability contributes to technological advances, because it provides troubleshooting information which a researcher can gain insight into. In my summarizer, it is possible to attribute an error to, for instance, the wrong word sense being chosen, or one interpretation of a proposition overpowering another interpretation.

In the following chapter, I will first review the state-of-the-art in summarization, such as summarizers based on word statistics, graphs of sentences, and rhetorical relations. Traditionally, general-purpose summarizers operate on relatively shallow representations. As there is little room for improvement using surface features, a recent trend of more sophisticated representations and processes which have characteristics of comprehension has emerged. I see my work as part of this development.

The chapter after this, Chapter 3, describes my summarization system, which is an implementation of the KvD model. This model was previously unimplemented in a form that fully operates on propositions as intended by KvD. It turns out that it needs many operational decisions when turning it fully automated. I will talk about how I operationalize by interpretation from their published papers and my own experimentation, and how I codify the intuitions behind their model. I also create a baseline model of comprehension, which is used to validate the key features of the KvD model, namely tree structures and incremental processing.

Chapter 4 describes my model of argument overlap. Argument overlap is modelled as continuous values, as opposed to the binary decisions which KvD used. Today's NLP offers many methods by which argument overlap could be realized. Out of these, my investigation leads me to investigate how coreference, distributional similarity, and lexical chains can be employed to arrive at an algorithmic model of argument overlap.

[2] `https://www.darpa.mil/program/explainable-artificial-intelligence`

On the basis of this, I realized several variants of my summarizer with different argument overlap capabilities, and with different output modalities. For instance, my summarizer can output extracted sentences, extracted word sequences, and in a collaborative work towards the end of my PhD research time, sentence material generated from propositions. These summarizers will be tested against various existing summarizers in Chapter 5, using both automatic and manual evaluation methods. The competing summarizers include state-of-the-art extractive and abstractive summarizers, as well as pipeline systems which combine extraction with compression.

In the final chapter, I will conclude this thesis and point to possible improvements.

Chapter 2

Background

The research I present in this thesis concerns the problem of automatic text summarization. In the literature, summarization systems are often distinguished on the following dimensions (Nenkova and McKeown, 2011):

Single-document or multi-document A single-document summarizer produces a summary of one document, whereas a multi-document summarizer can provide a gist of many documents of the same topic (such as news reports of the same event). Multi-document summarization has received considerable research attention in the recent years, not only because of its applications, but also because the additional complexity actually leads to more ways in which it can be improved, such as redundancy removal and information ordering, which are not as important in single-document summarization. Nonetheless, single-document summarization is still far from being solved, and has attracted the interest of researchers who want to apply deep, understanding-based models to summarization.

Generic, query-focused, or update Generic summarization assumes very little about the audience or the goal of the produced summary, and has to determine the intrinsic summary-worthiness of any piece of information. In contrast, query-focused summarization responds to a specific user query (such as producing snippets for search results), and update summarization only selects information that is new to the audience (such as the development of an event).

Extractive or abstractive An extractive summary only consists of sentences

taken unchanged from its original document, whereas an abstractive summary conveys information using newly generated language. Abstractive summarization has the advantages of being more natural and concise, as well as the possibility of presenting generalizations and inferred information. However, many shallow approaches which rank surface sentences directly are inherently incompatible with abstractive summarization,[1] and only models that operate on a meaning representation independent from the surface text have the potential of abstractive summarization.

The system I have created is a generic, single-document summarizer, which can produce both extractive and abstractive summaries. I choose this type of summarization because it is a natural application of a model of text comprehension, which can of course be extended for other types of summarization. Before I dive into the details of my system, I will first present an overview of the existing summarization techniques.

Each summarization system can involve a combination of many different technologies. In this chapter, the summarization methods are organized by their ways of representing the information content of documents. In Section 2.1, I will describe the traditional approaches, which mainly depend on statistical properties of words, lexical knowledge, and surface features, and which mostly produce extractive summaries. Summarization is regarded as a constrained optimization problem in which the selection of sentences is the variables. On the other hand, many researchers have explored the possibility of basing a summarizer on understanding, by first creating a structured representation of the meaning of a document, and then operate on that representation. Summarization is realized as different types of operations on the meaning representation. I will describe a few selected examples of systems which work in this way in Section 2.2.

2.1 Surface approaches

In this section, I will introduce summarizers which operate on relatively simple representations. The content of a document can be represented as a (weighted or unweighted) list of the most informative words (Subsection 2.1.1), a series of

[1] By applying post-processing such as sentence compression, extractive summaries can be converted into abstractive-looking ones.

topics (Subsection 2.1.2), or a graph of sentences (Subsection 2.1.3). To combine different indicators of sentence importance, machine learning can be applied (Subsection 2.1.4).

2.1.1 Frequency-based methods

The idea of determining the summary-worthiness of a word by its frequency can be traced back to Luhn (1958), who proposed the earliest method of automatic summarization. Luhn hypothesized that a word of high "resolving power" should be in the middle range of word frequencies in a document.

SumBasic (Nenkova and Vanderwende, 2005) is an extractive summarizer using only word frequencies of the input text. It first computes the probability of each word w_i as $p(w_i) = n/N$, where n is the number of occurrences of the word, and N is the number of all word occurrences. The score of a sentence is calculated as the average probability of the words it contains, and the highest scoring sentence is selected first for output. It then reduces the probability of every word in that sentence by squaring it, before selecting another sentence. This way of selecting sentence is an application of Maximal Marginal Relevance (MMR): it allows words with initially low probability to emerge as more influential later on, thus penalizing redundancy. The process of extraction and probability adjustment is repeated until the desired summary length is reached.

Instead of using word frequencies as they are, we can also normalize them in such a way that characteristic words of the document are given larger weights, while stop words (the most common words of a language) are given smaller weights. In information retrieval, this intuition is captured by the notion of tf–idf (term frequency–inverse document frequency), which is the product of two statistics: $\mathrm{tf}(w, d) \cdot \mathrm{idf}(w, D)$. Term frequency $\mathrm{tf}(w, d)$ is the frequency of the word w in the input document d. Inverse document frequency is:

$$\mathrm{idf}(w, D) = \log \frac{|D|}{|\{d \in D : w \in d\}|} \tag{2.1}$$

where $|D|$ is the number of documents of the background corpus D, and $|\{d \in D : w \in d\}|$ is the number of documents in D that contain the word w.

Word frequency and tf–idf are widely utilized in different types of summarizers. For example, Filatova and Hatzivassiloglou (2004) use tf–idf to weigh the content

units of a document, which they call "conceptual units". In their experiment, two types of conceptual units are used: individual words, as well as triplets consisting of two named entities and a verb-like connector. The distinction between conceptual units and textual units is a (albeit simple) reflection of the traditional idea of a meaning representation separate from the surface text. They regard each input sentence as a set consisting of conceptual units, and formulate summarization as a maximum set coverage problem, maximizing the total weight of covered conceptual units. Because the maximum set coverage problem is NP-hard, two greedy algorithms following the principle of MMR are used to select sentences iteratively.

In addition to tf–idf, there is an alternative way to find characteristic words of a document, namely the likelihood ratio (Dunning, 1993; Lin and Hovy, 2000). The likelihood ratio of a word $\lambda(w)$ is the ratio between the likelihoods of two competing hypotheses (denoting the input document as d, and the background corpus as D):[2]

H_1**:** $P(w|d) = p = P(w|D)$, i.e. the probability of w is the same in both corpora.

H_2**:** $P(w|d) = p_1 \neq p_2 = P(w|D)$, i.e. the presence of w is characteristic of d if $p_1 \gg p_2$.

The likelihoods of the two hypotheses can be computed by assuming that the frequency of a word follows the binomial distribution (where k represents the number of occurrences of w, and n represents the number of trials, i.e. the total number of word occurrences):

$$b(k; n, x) = \binom{n}{k} x^k (1-x)^{n-k} \tag{2.2}$$

$$L(H_1) = b(k_d; n_d, p) \cdot b(k_D; n_D, p) \tag{2.3}$$

$$L(H_2) = b(k_d; n_d, p_1) \cdot b(k_D; n_D, p_2) \tag{2.4}$$

The statistic $-2 \log \lambda(w)$ asymptotically has a χ^2 distribution. Therefore, a threshold of $-2 \log \lambda(w)$ can be determined by looking up a χ^2 distribution table for a particular confidence level (for instance, the threshold is 10.83 at 0.0001).

[2]Lin and Hovy formulate the hypotheses differently by using document probability given word instead of word probability given document. Here I use the formulation by Jurafsky and Martin (2009) and Nenkova and McKeown (2011) because it is more natural.

Those words which pass the threshold are called topic signatures by Lin and Hovy, as they are thought to reflect the topic of a document.

It has been shown, at least for query-focused multi-document summarization (Gupta et al., 2007), that binary word weights by thresholding the likelihood ratio outperform continuous weights. The topic signatures in a document can be imagined as a pseudo-sentence, which is the centroid of all sentences of that document. Many summarization systems, such as MEAD (Radev et al., 2004), use the overlap between a sentence and the centroid as an important indicator of the summary-worthiness of the sentence.

An entirely different way of using word probability is to apply the Kullback–Leibler (KL) divergence to summarization (Haghighi and Vanderwende, 2009). The KL divergence is a measure of the difference between two discrete probability distributions P_I and P_S:

$$D_{\mathrm{KL}}(P_I||P_S) = \sum_w P_I(w) \log \frac{P_I(w)}{P_S(w)} \tag{2.5}$$

where P_I and P_S are the word distributions of the input text and the summary, respectively. Summarization is done by minimizing the KL divergence of the summary from the input.

2.1.2 Topic-based methods

There are also summarizers which first detect the most important topics of a document, and then extract sentences to maximize the coverage of these topics. In contrast to the topic signatures, which represent *the* topic of a document using a group of words, these methods represent a document using a set of pre-defined topics. Each topic corresponds to a group of correlated words, or a distribution in which some correlated words are most frequent. Because of this, topic modelling can abstract away the variations of surface expression to certain extent, for example by regarding different terms as expressing the same topic.

Many topic models, such as lexical chains (Morris and Hirst, 1991) and latent semantic analysis (Landauer et al., 1998, LSA), have been applied to summarization. The topics of a document are either explicitly represented by lexical chains (equivalence classes of expressions which are considered to relate to the same concept), or implicitly by dimensions of a vector space (as in LSA). The

resulting summary consists of sentences which are presumed to be representative of the most salient topics.

I will revisit lexical chains and LSA in Chapter 4, and describe their algorithms there. However, rather than using themselves as representations of document content, I use them for a different purpose, i.e. to provide lexical knowledge to my model of text comprehension.

2.1.3 Graph-based methods

What I consider as a more advanced representation on which a summarizer can operate is a graph of sentences. In a graph representation, the content overlap between sentences is reflected by edges, and the importance of a sentence is determined by its relations with other sentences. In other words, the graph of sentences is a very basic model of the content structure of a document.

PageRank (Brin and Page, 1998) is an algorithm for computing eigenvector centrality based on the concept of random walk: Consider a surfer who begins at a node and randomly proceeds to another node following an edge, the importance of a node is reflected by the probability of the surfer visiting that node. A direct application of PageRank to sentence extraction is TextRank (Mihalcea and Tarau, 2004), in which the score of a sentence is recursively determined by the sentences similar to it. Unlike PageRank, which uses directed edges to represent hyperlinks from one webpage to another, the edges in TextRank are undirected. The weight of an edge is defined by the similarity between the pair of sentences it connects:

$$w_{ij} = w_{ji} = Similarity(S_i, S_j) = \frac{|S_i \cap S_j|}{\log |S_i| + \log |S_j|} \tag{2.6}$$

where S_i and S_j are sentences treated as sets of words. The scores of sentences are calculated iteratively. In iteration n, the score of a sentence S_i is updated based on the scores of its neighbours in the previous iteration:

$$Score_n(S_i) = (1-d) + d \cdot \sum_{S_j \in In(S_i)} \frac{w_{ji}}{\sum_{S_k \in Out(S_j)} w_{jk}} Score_{n-1}(S_j) \tag{2.7}$$

where d is the damping factor (which they set to 0.85). After the scores converge, the highest scoring sentences are extracted as the summary.

LexRank (Erkan and Radev, 2004) is similar to TextRank, the main difference

being that the computation of sentence similarity is based on tf–idf (tf is defined defined as the number of occurrences of a word in a sentence):

$$Similarity(S_i, S_j) = \frac{\sum_{w \in S_i, S_j} \mathrm{tf}_{w,S_i} \cdot \mathrm{tf}_{w,S_j} \cdot \mathrm{idf}_w^2}{\sqrt{\sum_{x \in S_i} (\mathrm{tf}_{x,S_i} \cdot \mathrm{idf}_x)^2} \cdot \sqrt{\sum_{y \in S_j} (\mathrm{tf}_{y,S_j} \cdot \mathrm{idf}_y)^2}} \tag{2.8}$$

where S_i and S_j are sentences treated as bags of words.

In the same paper, Erkan and Radev also present a degree centrality-based method, which performs similarly to eigenvector centrality and is computationally less expensive. However, because every edge of a node contributes equally to degree centrality, they have to choose a threshold of edge weight, and keep only the edges whose weights exceed the threshold.

As will become clear later, my summarization model contains a data structure that also connects nodes which represent content units. However, it is fundamentally different from the graph discussed here, because the data structure I use is incrementally updated to reflect the status of text processing (i.e. what is presumed to be the working memory at a particular moment of reading). In Chapter 5, I will compare the performance of my summarizer to that of TextRank and LexRank, as well as a graph centrality-based model which uses the same information as my summarizer.

2.1.4 Machine learning methods

Many supervised machine learning methods have been applied to the task of ranking or scoring sentences, such as decision trees (Hovy and Lin, 1998), maximum entropy (log-linear) models (Osborne, 2002), and support vector machines (Fuentes et al., 2007). The advantage of machine learning is that it can combine many different features, including discourse-based features such as cue phrase matching and sentence location, as well as frequency-based features (Kupiec et al., 1995; Teufel and Moens, 1997). Aone et al. (1997) additionally use knowledge-based features, i.e. word associations. MEAD is also an example of feature combiners, which includes among others both the centroid and LexRank as features.

Instead of evaluating each sentence in isolation, more sophisticated models search for a globally optimal way of extracting sentences. For instance, sentence selection is modelled as a sequence labelling problem in a hidden Markov

model (Conroy and O'leary, 2001) or a conditional random field (Shen et al., 2007). Integer linear programming (ILP) can also be used to maximize content coverage and minimize redundancy (McDonald, 2007; Galanis et al., 2012). These models allow features about summary coherence to be considered together with features for content selection, thus improving the text quality of the resulting summaries. To produce abstractive summaries, sentence extraction can also be jointed optimized with sentence compression, which is achieved by Nishikawa et al. (2014) using hidden semi-Markov models, and by Martins and Smith (2009) using ILP.

The stumbling block in improving supervised methods has always been the high cost of creating training data, or the lack thereof. This problem is also related to the automatic evaluation of summaries, i.e. how online feedback is provided to a machine learning system based on reference summaries, because human judgement cannot be obtained immediately and repeatedly. Therefore, semi-supervised learning (Wong et al., 2008), reinforcement learning (Rioux et al., 2014), creating negative examples (Fuentes et al., 2007), and bootstrapping (the idea of using one classifier's most confident examples to train another classifier) have been applied as a response to the data bottleneck.

2.2 Deep approaches

Many creative methods have been proposed to enable computers to summarize text in a human-like way. First, there is the traditional approach of using information extraction and template-based generation (Subsection 2.2.1). It is also possible to summarize based on a linguistic analysis of the structure of a document via discourse parsing (Subsection 2.2.2). Finally, there are models which represent the text understanding process rather than the document itself. One class of these models is psycholinguisticlly inspired, which is also the case for my summarizer (Subsection 2.2.3). The process in which neural networks generate abstracts can also be regarded as modelling the understanding process (Subsection 2.2.4).

2.2.1 Information extraction methods

An abstractive summary can be created by extracting key information from the input document. The FRUMP system (DeJong, 1982) implements this

idea by using a "sketchy script", which is a human-written specification of the expected components of a type of news events. A sketchy script can be activated automatically because words whose presumed senses match the script are detected, or because it is predicted by another script. The output is a one-sentence summary based on templates.

SUMMONS (Radev and McKeown, 1998) is an application of the same idea to multi-document summarization. In addition to extracting information according to scripts, it also identifies the relations among information from different sources, such as agreement, contradiction, generalization, and subsequent update. It uses many resources, including the CIA World Factbook and a news archive, to generate referring expressions of people and organizations.

A more advanced system of this kind is ABSUM (Genest and Lapalme, 2013), which creates predicate triples by pattern matching on a dependency parse. The pattern matching rules are defined by an abstraction scheme, which is associated with a generation template. Both the abstraction schemes and content selection are controlled by a manually-created task blueprint.

These methods are able to achieve very concise and high-quality summaries. Their process of summarization is easy to interpret. However, they all depend on hand-crafted knowledge representations for their target domains.

2.2.2 Discourse-based methods

Summarization can also be based on a discourse analysis according to Rhetorical Structure Theory (Mann and Thompson, 1988, RST), which represents the content of a document as a tree of elementary discourse units (EDUs). The EDUs correspond to sub-sentential spans of text, and are categorized into two types in an RST tree: nuclei, which are considered to convey important information, and satellites, which are considered be to supplementary to the nuclei. Intuitively, nuclei are favoured in summarization.

In Ono et al.'s (1994) system, each EDU is penalized by the number of satellites in the path from the root node of the RST tree to that node itself. It is later shown that rhetorical relations should also be considered in addition to nuclearity (Marcu, 1998). For example, satellites of certain types of ELABORATION relations are still summary-worthy, whereas satellites of an EXAMPLE relation are probably never important. In addition to heuristically scoring standard RST trees, it is also

possible to train a discourse parser specifically for summarization using summary data in place of RST annotation (Wang et al., 2015).

The RST-based methods and the topic-based methods of lexical chains and LSA are both ways of modelling coherence. The difference is that the former models rhetorical relations while the latter models lexical cohesion. Expectedly, RST-based methods work well in the genre of scientific articles (Teufel and Moens, 2002), in which rhetorical relations are often obvious. In contrast, lexical coherence is generally applicable to coherent texts of any genre, such as narratives. The RST tree should also be distinguished from the tree data structure in my system. Besides the difference that the RST tree is a static representation of an entire document, the reason under which the tree is constructed is also different, namely that my system attaches new nodes to the tree mainly by lexical cohesion. The idea of RST has overlap with the notion of macrostructure in the KvD model, which I will discuss in Chapter 3.

2.2.3 Psycholinguistic methods

My method of summarization is not alone. A model inspired by psycholinguistic theories does not only model the content of a document, but also models the process in which the content is understood and used. This explanatory and flexible approach towards summarization has shown its potential in the recent years.

Zhang et al. (2016) propose a summarizer based on the construction–integration model (Kintsch, 1988, CI model), which is a variant of the KvD model. The CI model also explains the text comprehension process using memory cycles. In a memory cycle, however, the reader's working memory is represented by a network in which words, phrases, concepts and propositions are nodes. This is different from the tree data structure I use, which only contains propositions.

The CI model explains the disambiguation of natural language as two phases of every memory cycle: In the construction phase, any concept or proposition associated with existing nodes could be added from the reader's long-term memory (i.e. knowledge) into the network; thus some nodes do not necessarily represent the correct interpretation. It is the responsibility of the integration phase, during which activation is spread from already-activated nodes to all nodes through the network, to filter out unwanted nodes by assigning low activation values to them. For example, let us consider a Winograd-style (1972) pronoun resolution problem.

In the text segment "*The lawyer discussed with the judge. He said 'I shall send the defendant to prison'*", the pronoun "*He*" could refer to the lawyer or the judge. But assuming that the template proposition SEND (JUDGE, DEFENDANT, PRISON) exists in the reader's long-term memory, the judge interpretation would overpower the lawyer interpretation.

As I have shown, the CI model alone can explain many NLP problems, which are currently studied as individual tasks. However, it has to assume a lot of knowledge. In their implementation, Zhang et al. use a network consisting only of words, without concepts or propositions. The connection strength between words is calculated using LSA. Although they have propositions (created in a similar way to my method), the propositions are only used in the generation stage for sub-sentential extraction. Therefore, their approach is in fact closer to a connectionist method of determining word importance than to my method, which is based on the manipulation of content units (i.e. propositions).

2.2.4 Neural network methods

In the most recent years, deep learning has become an attractive method for summarization, mainly owing to the very high level of data-drivenness that is achievable by sequence-to-sequence (seq2seq) models. A seq2seq model consists of an encoder, which encodes the meaning of the input text into a latent vector, as well as a decoder, which decodes the latent vector into textual summaries. The parameters of the encoder and the decoder are trained together using a large parallel corpus of text and summary pairs, which is usually the CNN / Daily Mail corpus (Hermann et al., 2015) consisting of 312k such pairs.

The most frequently used neural network architecture for sequential data is the recurrent neural network (RNN), which has several variants such as the long short-term memory (LSTM) and the gated recurrent unit (GRU). They are different from conventional feed-forward networks in that they maintain an internal state. At a particular time t, the internal state h_t is computed based on two inputs: the current input x_t and the previous internal state h_{t-1}. (For an encoder, x_t is a token of the input text; for a decoder at test time, x_t is o_{t-1}, the token that was output at the previous step.) The final state of the encoder becomes the initial state of the decoder. The general idea of recurrently updating a memory state and the mechanisms of forgetting and adding information bear

certain similarity to my approach. But in this case the memory state is represented as a vector, as opposed to the explicit graph structures used in my model. The use of a continuous vector space allows for efficient gradient-based learning.

Seq2seq models have been shown to provide high linguistic quality in sentence generation tasks such as sentence compression (Rush et al., 2015) and headline generation (Ayana et al., 2016). However, document summarization, which requires outputting multiple sentences that are informative of not only the overall topic but also a coherent set of key information bits, is still a challenge for these models (Yao et al., 2017). This is probably because the vector space alone can encode salient topics, but is insufficient to encode many concrete pieces of information and their relations at the same time. It is also hard to interpret the meaning of the values in these vectors, making the results less explainable.

To address the challenge of long inputs/outputs, the following mechanisms have been engineered for state-of-the-art seq2seq document summarizers such as Nallapati et al. (2016) and See et al. (2017):

Attention mechanism Using the attention mechanism (Bahdanau et al., 2015), the output of the decoder is computed based on not only the decoder's current state h_t^{dec} but also a weighted average of the encoder's state at every step h_i^{enc}:

$$o_t^{\text{dec}} = W_{\text{out}}\big[h_t^{\text{dec}}; \sum_i \alpha_{t,i} h_i^{\text{enc}}\big] \tag{2.9}$$

The attention weights $\alpha_{t,i}$ are computed from h_t^{dec} and h_i^{enc} (typically using a feed-forward network), and reflect how much the output at time t depends on the encoder state at time i. Thus, not just the final state of the encoder is used for decoding, but all intermediate states are accessible. Both Nallapati et al. and See et al. use the attention mechanism.

Pointer–generator network In addition to generating a word from a fixed vocabulary, the decoder at any time step can also copy a word directly from the input text using a pointer network (Vinyals et al., 2015). In a pointer network, the probability of outputting a word w is the sum of attention weights of all encoder states where w is the input token, i.e. $p_t^{\text{point}}(w) = \sum_{i:w_i=w} \alpha_{t,i}$. A soft switch is trained to compute a probability p_t^{gen}, meaning that with probability p_t^{gen} the generator using a fixed vocabulary is responsible for output at decoder step t, and with probability $1 - p_t^{\text{gen}}$ the pointer network is

responsible. This mechanism allows the decoder to strike a balance between extracting original content and generating novel expressions. Nallapati et al. use a pointer network for out-of-vocabulary words and named entities, whereas See et al. let the model learn when to use the pointer network and combine the results into a total output probability:

$$p_t(w) = p_t^{\text{gen}} p_t^{\text{vocab}}(w) + (1 - p_t^{\text{gen}}) p_t^{\text{point}}(w) \tag{2.10}$$

Coverage mechanism In order to reduce repetition in the output, which is a common problem for seq2seq models when generating multi-sentence texts, See et al. use the coverage mechanism (Tu et al., 2016) to distribute the attention weights more favourably to material that has not been covered so far. A coverage vector is computed as the sum of all attention vectors that were used in previous decoding steps, i.e. $c_{t,i} = \sum_{t'=0}^{t-1} \alpha_{t',i}$, and is used as an additional input for computing the attention vector of the current step α_t. This coverage information is incorporated into the loss function, essentially penalizing any α_t that is similar to c_t.

In addition to fully seq2seq models, other neural network architectures have been developed for the less challenging problem of extractive summarization. Two notable systems are Cheng and Lapata (2016) and Nallapati et al. (2017). Cheng and Lapata use a convolutional neural network to compute vector representations of sentences from word embeddings, and then use an RNN to compute the vector representation of the document from the sentence vectors. To produce a summary, they use an attentional LSTM decoder to either extract sentences or extract words. Nallapati et al. use RNNs for encoding both sentences from words and the document from sentences. To extract sentences, they use a classifier which computes the extraction probability of each sentence, given the document vector. The performance of these systems is better than or at least comparable to the abstractive seq2seq models, reflecting room for improvement in neural abstractive summarization.

In Chapter 5, I will compare the performance of my abstractive system to that of See et al. (2017), and my extractive system to Nallapati et al. (2017). Both these systems represent the state-of-the-art in supervised summarization.

Chapter 3

Simulation of comprehension

The main methodology of my summarization system is a simulation of human's text comprehension. Summarization in this way is explanatory because the summary-worthiness of a piece of information is determined by the memory operations it has undergone during the simulation. In this chapter, I will provide the framework of this simulation.

The general principles of the simulation are inspired by KvD, who propose a model that explains the comprehension process using a semantic representation consisting of content units called *propositions*. Each proposition is a small unit of information that can be individually selected for memory operations such as recall or summarization. KvD's specifications of proposition, and my way of creating propositions of a similar size will be discussed in Section 3.1.

The KvD model (Section 3.2) is mainly driven by the manipulation of propositions, which is influenced by local and global coherence, i.e. connecting propositions which concern the same concept, and organizing, creating, and changing propositions according to the goal of reading. The fact that the KvD model can predict the probability of recalling a proposition after reading makes summarization a natural test bed for it.

Many aspects of the KvD model have to change when we go from manual simulation to full automation. The main difference is that, in manual simulation an oracle provides a binary answer to complex questions, whereas in my implementation this oracle has to be broken down into small, statistical individual factors. When there are different interpretations, which are reflected by a *competition* of different possible ways of processing, these factors are considered together to

decide which memory operation is carried out. The memory operation which wins a competition is presumed to correspond to the interpretation that would arise into consciousness for a human reader, while alternative interpretations would be instinctively suppressed.

In Section 3.3, I will describe my model, which uses this notion of competition. The simulation takes as input a chunk of propositions at a time, and repeatedly updates a data structure representing a human's working memory. This data structure is a tree consisting of propositions connected by local coherence, and hence is called the memory tree. Competition is used in many processing stages of the model, such as when a new proposition is integrated into the memory tree, and when the memory tree is adjusted to reflect topic changes. The competitions are based on the output of NLP modules, which will be further discussed in Chapter 4, as well as factors about the status of a proposition in the memory tree. Finally, in Section 3.4, I will present a baseline model which uses the same information but does not perform such simulation. My model will be compared against this baseline in the experiments.

I will now turn to the building blocks of the simulation: propositions.

3.1 Propositions

While there are techniques such as sentence compression and paraphrasing that can make an extracted summary appear to be non-extractive, a truly abstractive summarizer must select content based on an internal semantic representation. Ideally, text information is processed into logical forms, abstracting away from different linguistic realizations. There are many semantic formalisms, some having implemented parsers and generators, that can potentially serve this purpose. The English Resource Grammar (ERG; Flickinger, 2000), which uses Minimal Recursion Semantics (MRS; Copestake et al., 2005) as its semantic representation, is an example. A relatively new semantic representation, Abstract Meaning Representation (AMR; Banarescu et al., 2013), is on a level of abstraction that arguably surpasses that of the ERG, but text generation from AMR was not possible until Flanigan et al. (2016).

The proposition is the basic unit of information selection in my summarizer. A proposition contains a predicate and one or more arguments, and is denoted in the

form of PREDICATE (ARGUMENT$_1$, ARGUMENT$_2$, ...) in this thesis. The notation is for the sake of presentation; in the implementation, a proposition is a data structure that supports two operations: the computation of overlap with another proposition, and the realization as a linguistic sentence. The fact that propositions are individually interpretable as short sentences is important, because it means the ranking of these units is not a mere modelling of word salience, but is directly associated with the inclusion or exclusion of content units that are meaningful for human to understand summaries. Because the graph nodes in Dependency MRS (DMRS) and AMR are word-level concepts, propositions correspond to subgraphs in these representations.

The propositions in the KvD model are defined by Kintsch (1974) and Turner and Greene (1977). In contrast to the more linguistically oriented representations, which semanticists typically work with, their propositions represent information units that are assumed to exist in the human mind, and individuals may derive different propositions as a result of comprehending the same piece of text. Text comprehension is the process of assimilating the textual tokens with a reader's mental lexicon, which is part of one's personal knowledge. An individual's mental lexicon consists of *word concepts*, which are abstract entities that are associated with linguistic and non-linguistic properties, and can be expressed in the surface text as words or phrases. In a proposition, the predicate is a relational word concept, and the arguments can be word concepts or embedded propositions. The arguments have semantic roles such as agent, experiencer, and instrument. For example, the sentence "*Mary trusts John*" gives rise to the following proposition involving three word concepts (named arbitrarily for the sake of presentation):

$$\text{TRUST (MARY, JOHN)} \tag{3.1}$$

The sentence "*If Mary trusts John, she is a fool*" gives rise to two propositions embedded in an outer proposition:

$$\text{IF (TRUST (MARY, JOHN), FOOL (MARY))} \tag{3.2}$$

The propositions that are explicitly mentioned in the text are ordered according to the textual position of their predicates, and are processed in this order in the KvD model. In addition to the explicitly mentioned propositions, there are also

propositions that are a result of the reader's generalization and construction. They are an important source of macropropositions, which will be discussed in Subsection 3.2.2.

How much content is packaged in one proposition is a question important to summarization systems of which the content selection relies on predefined units. If a proposition contains too little information, selecting that proposition without also selecting propositions related to it risks damage to truth-preservation and grammaticality of the summary output. If a proposition contains too much information, however, the content selection will become less effective as the summarizer is not able to choose a smaller, possibly more appropriate, set of meaning units. Instead, it is forced to select the entire package. Ideally, whether an optional argument amounts to a separate proposition should depend on information-status (the given/new distinction; Allerton, 1978; Prince, 1992), i.e. whether the proposition without such an argument still has enough listener-new information. Consider the sentence "*Susan was working on the model*". We could treat "*on the model*" as an adjunct, which would result in two propositions:

$$\begin{aligned} \alpha = & \textsc{work}\ (\textsc{Susan}) \\ & \textsc{object}\ (\alpha,\ \textsc{model}) \end{aligned} \tag{3.3}$$

This interpretation would be acceptable only if proposition α is non-trivial to the reader, for instance because Susan previously refused to work. Otherwise, the object of WORK ("*the model*") should be an argument, and therefore all information of the sentence is packaged in one proposition: WORK (SUSAN, MODEL).

Automatically creating KvD's propositions from a text is impossible with today's NLP technologies, because the process is at a very high level of abstraction, which depends on tremendous amount of knowledge. For example, proposition (3.1) may be created in a reader's mind based on sentences such as "*Mary believes in John*", "*Mary has faith in John*", "*Mary thinks John is trustworthy*", or even "*Mary gives her heart to John*". The process would require abilities to recognize paraphrases and textual entailments, as well as to understand figurative language.

The approach I take is to first create propositions that are close to surface texts, and then delegate the question of concept identity to existing technologies such as coreference resolution, and lexical and distributional semantics. Breaking down sentences into proposition-like units is a task faced by many summarization

researchers. A commonly used method is to pack grammatical dependencies aiming for an intuitive size of information packages (Genest and Lapalme, 2011; Zhang et al., 2016; Li et al., 2016). Some summarizers such as Falke et al. (2017) rely on Open Information Extraction (Open IE) extractors, which often are similarly based on dependency parses, and the extracted units do not have a formal definition (Stanovsky and Dagan, 2016). Likewise, my propositions are also created from dependencies, namely the Stanford Dependencies (de Marneffe et al., 2006). For example, the sentence "*If Mary trusts John, she is a fool*" gives rise to the following propositions:

```
trust (Mary, John)                          (3.4)
fool (she)                                  (3.5)
if (trust (Mary, John), fool (she))         (3.6)
```

In this example, proposition (3.6) is constructed from an adverbial clause, and propositions (3.4) and (3.5), which are embedded as arguments in proposition (3.6), are constructed from subject–verb–complement relations. In contrast to proposition (3.2), the pronoun `she` is not substituted by `Mary`, as it is the job of the argument overlap module (Chapter 4) to detect co-referring or synonymous expressions. A detailed description of the proposition building algorithm is provided in Appendix A.

As a data structure, the propositions in my implementation keep track of the textual tokens which give rise to them. These tokens are a baseline of text generation from the proposition, which is also useful for automatic evaluation. Admittedly, this baseline does not always produce truthful and grammatical realizations of the selected propositions. To produce high-quality abstractive summaries, my system is also run together with a language generation module based on the ERG. Details of this experiment will be presented in Section 5.4.

In the next section, I will present how propositions are manipulated according to KvD. After that, I will dive into the backbone of my processing model.

3.2 The KvD model

KvD present a model of human processing of text or speech, and make verifiable

predications about the content that will later be recalled by human subjects. The model has been validated by experiments in which the output of human subjects performing reading tasks is analysed in terms of propositions. It has been very influential, particularly in the 1980s and 1990s in educational (Palinscar and Brown, 1984; King, 1992) and cognitive (Paivio, 1990) psychology, and is still used today as a theoretical model of reading and comprehension (Zwaan, 2003; DeLong et al., 2005; Baddeley, 2007). Although it is mostly cited for the psycholinguistic concept of macrostructure (which I will introduce in the paragraph after next), the model also has an algorithmic aspect, which was temporarily overlooked due to the underdeveloped NLP technologies at that time.

KvD characterize the semantic structure of a discourse at two levels, namely the *microstructure* and the *macrostructure*. The microstructure consists of individual propositions, which are connected to one another by local coherence criteria. The most important single criterion for local coherence is referential coherence, realized as the same concept appearing in different propositions. Using referential coherence, KvD create a hierarchy of propositions called the *coherence graph*, in which the proposition which introduces a referent is superordinate to other propositions that also involve the same referent.

In contrast to microstructure, which reflects local coherence, macrostructure is about the organization of the entire discourse. The construction of macrostructure is controlled by the reader's goals in reading. It is well-known that knowledge of macrostructure can be used for constructing summaries, if all texts to be summarized come from the same text genre, and if the macrostructure of that genre is known. I have mentioned a few abstractive summarization systems which are based on filling slots of information in Chapter 2, such as FRUMP (DeJong, 1982) and SUMMONS (Radev and McKeown, 1998).

Early proposals of summarization methods influenced by KvD, including Correira (1980) and Hahn and Reimer (1984), are also primarily concerned with inferring the macrostructure of an entire document using world knowledge. The SUSY system (Fum et al., 1982) is slightly different in that it has an internal representation that is somewhat similar to microstructure. Fum et al.'s decision reflects an interest in the role of local coherence in summarization. In these approaches, the resulting summary can be explained by the structure of their internal representations. My goal is to follow in the footstep of these explanatory approaches, while using today's NLP technologies to make them more robust.

My summarizer is based on local coherence, which is generally applicable in any domain or genre. In addition to the use of probabilistic argument overlap, it has another important difference from SUSY: It models comprehension incrementally, not statically, in the sense that the memory state is changed each time a new sentence is processed, rather than once for the entire text. This also provides more informative and more fine-grained explanation for the resulting summary than a static structure, because one can inspect how a piece of information is first connected to existing knowledge, when it is moved to a higher status, and when it is forgotten or recalled. My model does not currently contain a simulation of the macrostructure, which is still open to future explorations. I will revisit the problem of macrostructure in Subsection 3.2.2.

KvD base their processing on the assumption of human memory limitations. This assumption states that human working memory can hold a limited number of propositions. As a consequence, the propositions of an entire discourse have to be processed incrementally. The reading process is broken down into individual *memory cycles*. In each cycle, new propositions are added to the working memory, and existing propositions which are predicted to be unnecessary for understanding the remaining content are forgotten. In the following subsection, I will describe the mechanism of the memory cycle.

Hierarchical structure and incremental processing are two essential features of the KvD model. My simulation of text comprehension also has these features; in fact, these constitute the main influence of the KvD model on my work.

Of course, it is possible that these two features are not, after all, necessary to model coherence, or that they are actually a hindrance. For instance, it might be possible that a graph containing all propositions and their coherence relations works equally well. This is a testable hypothesis, for which I have created a baseline system (cf. Section 3.4).

3.2.1 Memory cycles

A memory cycle is a procedure in which the working memory is updated using new information as the reading proceeds. Figure 3.1 illustrates the memory cycle in relation to the input/output modules on the left side; the three boxes inside the memory cycle represent different states of the working memory. At the beginning of a memory cycle, the working memory contains propositions that have been

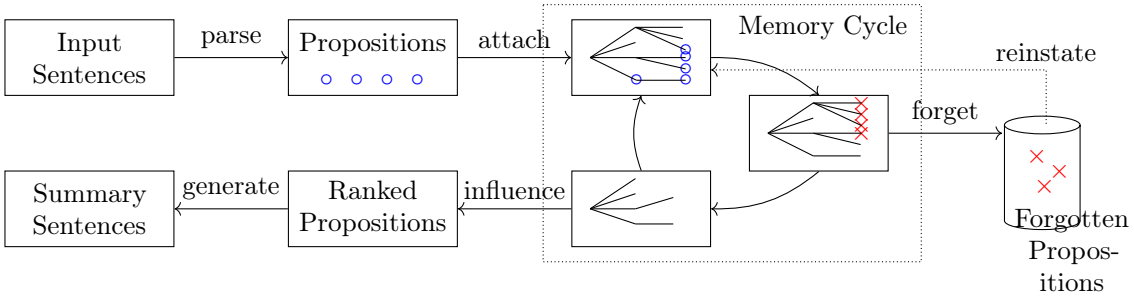


Figure 3.1: The memory cycles

retained from the previous cycle. Then, a chunk of new propositions (represented by ∘) have to be "understood" on the basis of these existing propositions, i.e. they have to be connected (directly or via intermediate propositions) to these propositions. Once the new propositions have been added to the coherence graph, a selection algorithm is performed on the graph in order to retain only a fixed number of propositions for the next cycle. Propositions that are not selected are considered forgotten (represented by ×), and will only be reinstated in a later cycle under special circumstances.

KvD conceive a way to calculate the probability of a proposition being regenerated in an after-reading task based on its participation in memory cycles. They assume that in each cycle, a proposition in the working memory may be chosen for later reproduction with probability s, which is a model parameter. In a recall or summarization task, a proposition will be reproduced if it is chosen in any cycle. Assuming that choosing a proposition for reproduction in one cycle and in another cycle are independent random events, if a proposition has participated in n cycles (i.e. being selected $n-1$ times to be retained in memory), the probability of reproducing it would be $1-(1-s)^n$. This means a proposition is more likely to be reproduced in a summary if it participates in more cycles.

There are two model parameters that control the simulated memory capacity. The number of new propositions to be processed in a cycle is influenced by surface clues such as sentence breaks, and is bounded by the maximum chunk size, which is a model parameter depending on the text and the reader. Typically, one chunk of propositions corresponds to one sentence. The number of propositions to retain in memory at the end of a cycle is also a model parameter.

KvD demonstrate a manual simulation of the memory cycles using the first paragraph of a research report by Heussenstamm (1971), titled *Bumper Stickers*

1	A series of violent, bloody encounters between police and Black Panther Party members punctuated the early summer days of 1969.
2	Soon after, a group of Black students I teach at California State College, Los Angeles, who were members of the Panther Party, began to complain of continuous harassment by law enforcement officers.
3	Among their many grievances, they complained about receiving so many traffic citations that some were in danger of losing their driving privileges.
4	During one lengthy discussion, we realized that all of them drove automobiles with Panther Party signs glued to their bumpers.
5	This is a report of a study that I undertook to assess the seriousness of their charges and to determine whether we were hearing the voice of paranoia or reality.

Table 3.1: Sentences of the *Bumper Stickers and the Cops* text

and the Cops. Its content is shown in Table 3.1, where each sentence corresponds to one memory cycle in the demo. Many algorithmic properties of the memory cycles are not explicitly specified by KvD. For anyone who attempts an implementation of the model, this example is the only source of information about the details intended by KvD, and this is the route I adopt here.

In the following sub-subsections, I will analyse the mechanisms of (1) the attaching of new propositions to the coherence graph, (2) the forgetting of old propositions, (3) the reinstating of forgotten propositions to the coherence graph, and (4) the handling of incoherence, mainly by drawing on the first, the second, and the last memory cycles of this example.

3.2.1.1 Attaching new propositions

The first cycle begins with an empty working memory. Therefore, the coherence graph that results from this cycle will contain only the new propositions (those corresponding to the first sentence). The propositions of the first sentence and the resulting coherence graph are shown in Figure 3.2.

My first observation is that a coherence graphs is a tree. Intuitively, the root of the tree should be the most important and general proposition of the document, so that most other propositions can be situated on relatively high levels (levels which are close to the root). The root can be seen as representing the current topic, its children and grandchildren representing different aspects of the topic and further specialized information. KvD only state that hierarchy of generality is a feature of macrostructure. However, I learn from the example that information generality

New propositions:

1. SERIES (ENCOUNTER)
2. VIOLENT (ENCOUNTER)
3. BLOODY (ENCOUNTER)
4. BETWEEN (ENCOUNTER, POLICE, BLACK PANTHER)
5. TIME: IN (ENCOUNTER, SUMMER)
6. EARLY (SUMMER)
7. TIME: IN (SUMMER, 1969)

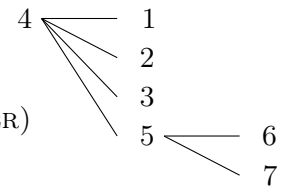


Figure 3.2: Cycle 1 of the *Bumper Stickers and the Cops* simulation

is also implicitly reflected in microstructure by the way how the coherence graph is constructed.

In this example, the root is determined by concept overlap with the title of the text: Proposition 4 is made the root of the tree because it shares the concept POLICE with the title. However, it is not clear how the root is determined in general. One method is to determine by node connectivity, i.e. choosing the root such that "the simplest coherence graph" results, which is proposed by KvD. The simplest tree can be defined as the one having the minimum average depth of nodes. Another method is to make the proposition which underlies the main clause of the first sentence the root (Kintsch and Vipond, 1979).

After placing proposition 4 in the coherence graph, propositions 1, 2, 3, and 5 can now attach under proposition 4 because they share the argument ENCOUNTER with it. Similarly, after proposition 5 is attached, propositions 6 and 7 can attach under it, because they share the argument SUMMER with it.

In a memory cycle, the attaching of new propositions can be interpreted as an iterative process. In each iteration, all new propositions which share any arguments with existing propositions on the tree are attached. Once a new proposition is attached, it itself becomes a possible site of attachment in the next iteration. The iterations continue until no more pending propositions can be attached.

To exemplify this, let us consider Cycle 2, in which propositions of the second sentence are to be attached to a tree consisting of the four old propositions (4, 5, 7, 3) retained from Cycle 1 (the reason why these four propositions are retained will be given in the next sub-subsection). Figure 3.3 shows the coherence graph after new propositions are attached, with the old subgraph in boldface. Among the new propositions, propositions 9, 15, and 19 are attached first, because they share arguments with proposition 4. The other new propositions have to wait,

Old propositions:

3. BLOODY (ENCOUNTER)
4. BETWEEN (ENCOUNTER, POLICE, BLACK PANTHER)
5. TIME: IN (ENCOUNTER, SUMMER)
7. TIME: IN (SUMMER, 1969)

New propositions:

8. SOON (#9)
9. AFTER (#4, #16)
10. GROUP (STUDENT)
11. BLACK (STUDENT)
12. TEACH (SPEAKER, STUDENT)
13. LOCATION: AT (#12, CAL STATE COLLEGE)
14. LOCATION: AT (CAL STATE COLLEGE, LA)
15. IS A (STUDENT, BLACK PANTHER)
16. BEGIN (#17)
17. COMPLAIN (STUDENT, #19)
18. CONTINUOUS (#19)
19. HARASS (POLICE, STUDENT)

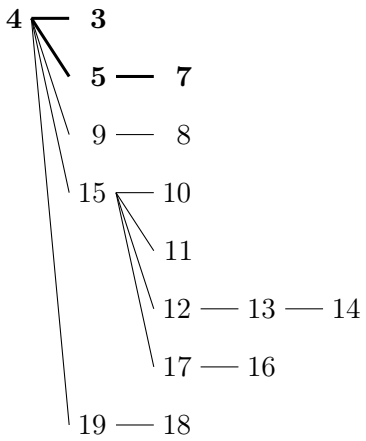


Figure 3.3: Cycle 2 of the *Bumper Stickers and the Cops* simulation

because they do not share any arguments with the four old propositions. But after propositions 9, 15, and 19 are attached, these propositions themselves become possible sites of attachment. Hence, propositions 8, 10, 11, 12, 17, and 18 can now be attached because they share arguments with propositions 9, 15, and 19. Next, propositions 13 and 16 are attached, and finally proposition 14 also finds its place.

Additional complexity ensues if we consider how embedded propositions are treated. As I have explained in Section 3.1, a proposition can be embedded in another proposition as an argument, which I represent using the # sign followed by its proposition number. A proposition can be attached under an embedded proposition it contains. For example, proposition 8 is attached under 9, which is embedded in 8 because proposition 8 provides additional detail about 9. Unlike attaching by argument sharing, attaching by embedded proposition is directional, i.e. a proposition is not normally attached under another proposition which contains it as an embedded proposition. This rule explains why proposition 16 is not attached under 9, which contains 16 as an argument. The phenomenon of attaching proposition 16 under 17 cannot be explained by the order of attachments, because proposition 9 has already become an available site of attachment before 17 becomes available.

3.2.1.2 Forgetting propositions

Because of the assumed limited capacity of the working memory, only some propositions can remain in memory before the next cycle starts. As I have explained in the previous sub-subsection, information specificity increases with tree levels. As general statements are more likely to be useful for understanding the rest of the text, a predictive algorithm should therefore prefer to retain propositions on high tree levels. The principle of retaining high-level propositions is also supported by Kintsch's (1974) experimental finding that a superordinate proposition in the coherence graph is recalled two or three times more often than its subordinate propositions by a human reader.

The memory selection algorithm used by KvD is called the leading-edge strategy. It is a combination of recency and generality requirements, which correspond respectively to a depth-first stage and a breadth-first stage. In the first stage, which is depth-first, propositions are selected following the "lower edge" of the graph, i.e. the most recent child nodes are picked up in depth-first order. Because propositions are ordered by their occurrence in the input text, a higher proposition number corresponds to a more recent proposition. For Cycle 1, the lower edge consists of propositions 4, 5, and 7. There is an additional restriction that each child node to be picked up must be more recent than its parent. Therefore, for Cycle 2, the lower edge consists only of 4 and 19, because 18 is older than 19, and the lower edge is considered to have ended here. If the memory capacity has not become full, the second stage is carried out to select additional propositions in breadth-first order, i.e. level by level. Propositions on the same level are picked in sequence, from the most recent one to the least recent one. In Cycle 1, additional propositions would be selected in the order 3, 2, 1, 7, 6. The algorithm stops when the number of selected propositions reaches the memory capacity. For the example demonstration in their paper, KvD set the memory capacity at four propositions. Hence, at the end of Cycle 1, propositions 4, 5, 7, and 3 are retained, while the other propositions are pruned.

3.2.1.3 Reinstating forgotten propositions

During the process of attaching new propositions to the coherence graph, if at any stage no connection is possible between the set of propositions to be attached and the set of nodes of the coherence graph, a proposition has to be found from

Old propositions:

4. BETWEEN (ENCOUNTER, POLICE, BLACK PANTHER)
19. HARASS (POLICE, STUDENT)
36. BLACK PANTHER (SIGN)
37. GLUED (SIGN, BUMPER)

New propositions:

38. REPORT (SPEAKER, STUDY)
39. DO (SPEAKER, STUDY)
40. PURPOSE (STUDY, #41)
41. ASSESS (STUDY, #42, #43)
42. TRUE (#17)
43. HEAR (#31, #44)
44. OR (#45, #46)
45. OF REALITY (VOICE)
46. OF PARANOIA (VOICE)

Reinstated forgotten propositions:

17. COMPLAIN (STUDENT, #19)
31. AND (STUDENT, SPEAKER)

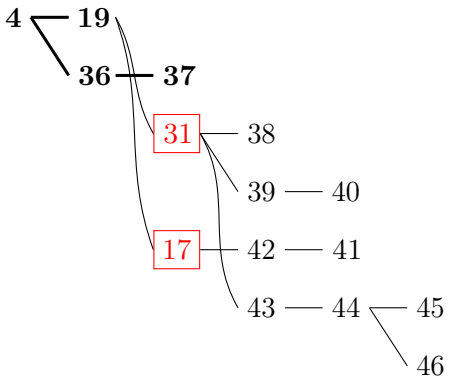


Figure 3.4: Cycle 5 of the *Bumper Stickers and the Cops* simulation

somewhere else to bridge the two sets. One possible source of bridging propositions is the set of previously forgotten propositions. Although forgotten propositions are technically speaking no longer available, KvD assume that humans can – at further cognitive cost – somehow recall them from the previously read content, and "reinstate" them into the working memory.

After four sentences are processed, Cycle 5 of the demonstration contains such an example (Figure 3.4). At the beginning of the cycle, none of the new propositions (38–46) share any argument with the existing propositions (4, 19, 36, 37), and therefore no proposition can be attached. KvD reinstate propositions 17 and 31 (highlighted), because these propositions can provide a connection to propositions 42 and 43 respectively. After the reinstatement, all new propositions can now be added to the coherence graph.

KvD search for propositions to reinstate by recursively tracing embedded propositions in any proposition that currently cannot to be attached. In this example, propositions 17 and 31 are considered for reinstatement because they are embedded in propositions 42 and 43. KvD regard the recalling of propositions 17 and 31 as reflecting the cognitive process of resolving the discourse referents of "*their charges*" and "*we*" to these two propositions.

There are two subtleties reflected by this example of reinstatement. First,

when reinstated, a proposition can attach under a node other than its original attachment site. Both propositions 17 and 31 used to be children of proposition 15, which was where they were added to the tree as new propositions in Cycle 2 and 4. But proposition 15 is no longer contained in the current tree. Luckily, they can now become children of proposition 19, which also shares arguments with them. As a general rule, KvD reinstate as few propositions as necessary. For example, if proposition 19 was also forgotten, reinstating propositions 17 and 31 would not have been enough because they do not have a connection with the tree nodes either. In this case, KvD would have also reinstated proposition 19 because it is embedded in proposition 17.

The second subtlety concerns the definition of proposition recency. In the graphical layout, KvD order the propositions on each tree level by their recency, so that the lower edge of the coherence graph corresponds to the path from the most general information (the root proposition) to the most recent information (the latest proposition). However, they place the reinstated propositions 31 and 17 below 37, which is more recent in terms of pure text order. This means they consider reinstated propositions as more recent than any proposition attached before the current cycle, i.e. the recency of a proposition is renewed when it is reinstated. Although KvD do not show the result of the leading-edge strategy at the end of Cycle 5, following the deduced meaning of the graphical layout, propositions 4, 19, 31, and 43 would have been retained in memory by the strategy.

3.2.1.4 Handling incoherence

Reinstating a forgotten proposition is one of the strategies to handle incoherence. Incoherence can happen during a simulated comprehension process due to various possible reasons. For example, it may be due to the memory selection algorithm (the leading-edge strategy) failing to select useful propositions in a previous cycle, or due to a memory capacity that is too small for the text being read. It may also be a result of the imperfect fluency of the text itself. In theory, if a reader's way of comprehension is modelled perfectly, occurrences of incoherence is predictive of difficulties in comprehension.

In addition to forgotten propositions, propositions which exist in the reader's long-term memory (representing the knowledge of the reader) are also used to connect input propositions which appear to be disconnected. Because a reader

can use inference to supply the missing links to make the discourse coherent, writers can leave all propositions implicit which they assume can be inferred by the readers, such as presuppositions. This is codified in pragmatics as the Grice's principles (Grice, 1975). The propositions shown by KvD can be regarded as a result after inference has been applied in such a way that coherence can be established by the repetition of symbolic concepts.

Because KvD assume coherent input text, all propositions should eventually be connected, occasionally with the help of reinstatement. An example of true incoherence is discussed by Kintsch and Vipond (1979). In such a scenario, there are propositions from the text which are considered to be completely disconnected. One or more separate trees have to be created to provide space for such remaining propositions. As a result, the coherence graph becomes a forest of disjoint trees, reflecting the incoherence of the discourse. Although Kintsch and Vipond do not discuss what will happen next, I assume that the model should allow a new proposition in a later cycle to connect the disjoint trees, reattaining coherence.

3.2.2 Macro-operations

Up to now, I have discussed microstructure (i.e. the coherence graph), but in the KvD model there is also a macrostructure. Like microstructure, the macrostructure is also a tree, but it captures the global coherence of a discourse. The macrostructure tree is organized by generality and relation to the goal of reading. At the top level, the information of an entire discourse is reduced to only one proposition. In the *Bumper Stickers and the Cops* example, the root proposition of the macrostructure is DO (EXPERIMENT), i.e. an experiment was done. One of the child nodes of the root is a node for the experiment setting, which is occupied by a proposition denoting that the experiment was done in California. The children of this node convey further details about the setting, such as the college (California State College) and the time (1960s).

The construction of macrostructure is controlled by the *schema*, the formal representation of the reader's goals. KvD limit their discussion to reading tasks that have clear goals, hence a comprehension process is always associated with a schema. In a comprehension experiment, a schema can be enforced on all human subjects, by either choosing reading material that belongs to an established genre (such as stories and psychological research reports), or requiring a specific purpose

of the reading (such as reading a review of a play in order to decide whether to go to that play). If a loosely structured text is read with no clear goals, KvD argue that the resulting macrostructure is, in principle, indeterminate.

Macrostructure is constructed from the bottom up using macro-operations. A macro-operation is a transformation that maps one or more propositions in lower levels of the macrostructure tree to zero or one proposition in a higher level of the same tree. Micropropositions, i.e. propositions in microstructure, can be regarded as situated below the bottom of macrostructure. To construct the first (lowest) level of macrostructure, micropropositions are processed by macro-operations, which may for example construct a macroproposition (a proposition in macrostructure) out of several micropropositions, or carry over a microproposition into macrostructure unchanged. The other levels of macrostructure are similarly constructed from propositions in lower levels. Macrostructure only contains propositions that are relevant to the schema, and the criterion for schema relevance is stricter in higher levels. Hence, only the propositions which answer the most important questions of the schema, and which are typically more general than others, can arrive at a high level.

Macro-operations can be classified by the relation between their input from lower levels and output in a higher level. There are four types of macro-operations:

Deletion: A proposition is not included in the output level if it is considered irrelevant to the schema at that level.

Carrying over: A proposition can be carried over unchanged into the output level if it is considered relevant. A proposition can be both a micro- and a macroproposition at the same time, or a macroproposition on different levels. For example, microproposition 40 (Figure 3.4) is also a macroproposition because it states the purpose of the study, which is a slot in the supposed "research report" schema.

Generalization: A generalized proposition can be created by generalizing the information of one or more propositions (micro- or macro-). KvD consider generalization as a phenomenon independent of the schema. Only those generalized propositions which are relevant to the schema are stored in the output level. For example, proposition 1 (Figure 3.2) can be generalized to SOME (ENCOUNTER), and propositions 5 and 7 (Figure 3.2) together can be

generalized to TIME: IN (EPISODE, 1960S). The latter fills the "time" slot of the schema, and therefore is a macroproposition; whereas the former is not.

Construction: A constructed proposition can be created on the basis of one or more propositions (micro- or macro-). Similar to generalization, a constructed proposition is included in the output level only if it is relevant. A construction can denote a global fact which the propositions giving rise to are normal conditions, components, or consequences. For example, a construction of "eating in a restaurant" may arise from a sequence of propositions denoting events such as ordering food and paying.

Like in microstructure, propositions in macrostructure are also chosen probabilistically for later reproduction, such as in a summary. Let us denote the probabilistic trial of marking (with probability s, which I have defined in the previous subsection) a microproposition p for reproduction as $S(p)$, and that of marking (with probability m) a macroproposition p for reproduction as $M(p)$. In microstructure, if a proposition p participates in n memory cycles, $S(p)$ has happened n times. In macrostructure, however, the number of times $M(p)$ has happened is instead controlled by its position in the macrostructure tree. Because a macroproposition is considered to be promoted from the bottom level, and $M(p)$ is assumed to happen every time it is promoted, the number of occurrences of $M(p)$ equals to its level counted from the bottom (which is level 1). Therefore, the reproduction probability of a pure macroproposition (a macroproposition that is not a microproposition at the same time) situated on level n is $1-(1-m)^n$. KvD assume that the results of $S(p)$ and $M(p)$ are statistically independent. Hence the reproduction probability of a proposition that is both a micro- and a macroproposition can be similarly calculated.

In addition to micro- and macropropositions, two other types of propositions may also be reproduced: generalized propositions and constructed propositions, with probability g. Obviously, the model parameters s, m, and g dictate the relative weight of microstructure, macrostructure, and generalization/construction in a human reader's memory.

KvD conduct two experiments to investigate the influence of the type and the time of the after-reading task on the model parameters s, m, and g. They estimate these parameters by analysing texts output by human subjects into propositions, and aligning these propositions to their predicated propositions of all

three types. The first finding is that the probability s is comparatively lower if the task is summarization rather than recall, which is reflected by the relatively higher proportion of macropropositions, generalizations, and constructions in summaries. The second finding is that s and g dramatically decreases when human subjects are asked to recall after 3 months rather than immediately, while m changes very little over time. This means that macropropositions are usually remembered longer than other propositions.

These findings tell us that macropropositions are important to summarization. Therefore, for the sake of abstractive summarization, it would be attractive to simulate macrostructure in addition to microstructure. (KvD are able to create their macropropositions without problem because they rely on human intuition.) However, as soon as we try to automate this procedure, robust generation of inferred propositions would require the use of world and contextual knowledge and inference, which is beyond the scope of current NLP technologies. A related area of research is the automatic inference of discourse structure, by supervised (cf. Subsection 2.2.2) or unsupervised (Liu and Lapata, 2018) learning, but it is only concerned with generalizations that already exist in the input text, and is nonetheless incapable of creating new propositions.

KvD's separation of the micro-operations from the macro-operations is important, as it allows me to simulate one module of the comprehension process (micro-operations) computationally, and identify components that are not currently manageable. In my summarization system, I manipulate propositions by using the micro-operations and some other operations I have devised. It is scientifically interesting to test how far a relatively simple model can take us in the way towards a simulation of comprehension that has practical application, even in our current situation with limited technologies for modelling knowledge or inference in natural language.

3.3 My model of comprehension

I will now present my adaptation of the KvD model and the resulting implementation. First of all, at the core of my model is also a tree of propositions corresponding to the coherence graph, which I call the *memory tree*. The main difference from the KvD model is that different ways of attaching and reinstating

propositions are compared numerically.

The question I will have to solve is how to choose an attachment site if multiple possible sites exist. KvD do not explicitly answer this question. In fact, in their manual, idealized simulation, such situations rarely occur, so I cannot solve this question by observation from KvD. Instead, I have to create my own means of evaluating different options of attachment. In my model, three factors affect the quality of an attachment site for a given proposition: (1) the strength of argument overlap between the two propositions, (2) the location of the attachment site in the memory tree, and (3) the number of forgotten propositions that need to be reinstated in order to make the attachment site available for attachment. I will now discuss these factors in turn.

Strength of overlap: KvD use a binary notion of overlap, but I use a continuous one. In KvD's simulation, the arguments are discrete concepts, which only overlap with identical concepts. A lot of intellectual work goes into creating these concepts in the first place, but in KvD's demonstration, this work remains invisible as it is done during proposition building, which in their simulated case has access to world knowledge and reasoning. In contrast, when operationalizing the model like I do, this work has to be made explicit, and it takes place after proposition building. In my model, the overlap of two arguments (taken from their respective propositions) is modelled continuously, i.e. as a real number between 0 and 1. How this strength of argument overlap is computed will be discussed in its own chapter, Chapter 4.

Memory location: There are two reasons, both originating from what I have described in the previous section, why attaching on a higher level of the memory tree should be preferred. First, KvD would first attach all propositions that can be attached, and then use the newly attached propositions to connect to more propositions. A proposition is attached lower in the tree only if it cannot connect directly to higher-level propositions and requires other propositions in the intermediate levels to provide the connection. Second, the tree is structured in such a way that different subtrees under the same node represent specialized content in different directions. By carefully controlling the weight of the location factor, propositions about the same topic can be clustered together.

Memory status: A major difference between my memory tree and KvD's coherence graph is that I keep all propositions ever processed, whereas KvD describe "forgotten" propositions as being entirely removed from the coherence graph. For each proposition in a memory tree, I introduce an attribute called the memory status, which can be either activated or deactivated (i.e. temporarily forgotten). At the end of each cycle, I select the propositions that are to remain activated for the next cycle, in such a way that the subgraph consisting of only activated propositions is also a tree (henceforth the subgraph connectivity criterion). Such a subgraph would represent the working memory in a way that is similar to the coherence graphs in Figures 3.2, 3.3, and 3.4. Because a newly attached proposition is by definition activated, if it is attached under a deactivated node, one or more propositions need to be reactivated (reinstated) for the subgraph connectivity criterion. Besides other solutions which I will discuss later, reactivating every deactivated ancestor is a guaranteed solution. Therefore, the number of deactivated ancestors (including the attachment site itself) is an upper bound of the number of propositions to reinstate. Of course, reinstating fewer propositions is more favourable.

Why does the attachment site of a new proposition need to be selected based on the combination of the three factors, instead of any single factor? When deciding the attachment site of a proposition p, there is often a choice between attaching using a weak overlap under a site that is salient in memory (having a favourable memory location/status), and attaching under a less salient site but which has a stronger overlap with p. If this problem can be solved by human cognition, then the attachment site that wins the competition corresponds to the most obvious choice in context, whereas the other sites are instinctively suppressed. If we choose the attachment site only by the strength of overlap, we would have essentially limited attachment to propositions which originated from the same sentence, because they share identical word tokens (overlap = 1). Coherence across sentences can be achieved by regarding reasonably strong overlaps as (almost) equally good, and letting the memory location/status influence the decision. If we place too much weight on memory salience, many attachments may be based on the noise in the argument overlap model we use, which may result in a small but non-zero strength of overlap for any arbitrary pair of words.

Algorithm 1 The outline of the memory cycles. The subroutine AddPropositions will be defined in Algorithm 3 (page 49). The subroutines InsertRoot, StartAlternativeTree, MemorySelect, and AdjustRoot are explained later in text; their pseudocode is in Appendix B.

Require: *propositionChunks*, a list of chunks, each being a list of propositions
Require: *tree*, an empty memory tree, in which nodes are propositions
Ensure: the *tree* reflects the final status after all *propositionChunks* are processed

```
 1: procedure RunMemoryCycles(propositionChunks, tree)
 2:     pendingPropositions ← ∅
 3:     for all chunk ∈ propositionChunks do
 4:         pendingPropositions ← pendingPropositions ∪ chunk
 5:         if tree is empty then
 6:             InsertRoot(pendingPropositions, tree)
 7:         end if
 8:         AddPropositions(pendingPropositions, tree)
 9:         if pendingPropositions has more propositions than tree then
10:             restarted ← StartAlternativeTree(pendingPropositions, tree)
11:             if restarted then
12:                 AddPropositions(pendingPropositions, tree)
13:             end if
14:         end if
15:         MemorySelect(tree)
16:         rootAdjustable ← True
17:         while rootAdjustable do
18:             rootAdjustable ← AdjustRoot(tree)
19:         end while
20:     end for
21: end procedure
```

The outline of my simulation of text comprehension is provided in Algorithm 1. Each iteration corresponds to a memory cycle, in which a chunk of propositions is added to the list of pending propositions and then processed. In the beginning of the first cycle, one of the pending propositions is selected to be the root of the memory tree (Line 6). Pending propositions are added to the tree on Line 8, under competition between attachment sites. Before and after selecting activated propositions for the next cycle (Line 15), I perform two adjustments, namely the recreation of the tree in the unlikely case of failing to attach most of the propositions, and the change of tree root if a subtree becomes dominant (cf. Subsection 3.3.2). I will now describe the process of inserting propositions to the memory tree, which corresponds to the subroutine AddPropositions.

Algorithm 2 Helper function and data structure for Algorithm 3 and 4.

```
 1: structure AttachmentPlan
 2:     node: a proposition
 3:     site: a proposition
 4:     transplant: an AttachmentPlan or null
 5:     score: a real number
 6: end structure
Require: plan, an attachment plan
Require: tree, a memory tree
Ensure: the changes specified in plan are executed on tree, and the minimal subtree
    that is affected by these changes is returned
 7: function ExecuteAttachment(plan, tree)
 8:     if plan.transplant exists then
 9:         detach plan.transplant.node from tree
10:         attach plan.transplant.node as a child of plan.transplant.site in tree
11:     end if
12:     attach plan.node as a child of plan.site in tree
13:     path ← the path from tree.root to plan.site
14:     if all nodes in path are activated then
15:         subtreeRoot ← plan.node
16:     else
17:         subtreeRoot ← the first deactivated node in path
18:     end if
19:     activate all nodes in path
20:     return the subtree of tree rooted at subtreeRoot
21: end function
```

3.3.1 Adding propositions

The process of adding propositions to the memory tree starts by determining the attachment plan for each pending proposition. An attachment plan (cf. Algorithm 2) specifies the particular site of attachment and other changes of the tree that will become necessary when the attachment plan is executed. An attachment plan is associated with a score. Then, there is an iteration of proposition insertion and plan updating. In each iteration, the attachment plan that has the highest score is executed, which results in the addition of one proposition to the tree. The attachment plans of the remaining propositions are then updated with respect to the changes to the tree.

This procedure is summarized as Algorithm 3. The return value of the function ExecuteAttachment (cf. Algorithm 2) is a pointer to the minimal subtree that is affected by the executed attachment. A subtree is affected if it contains

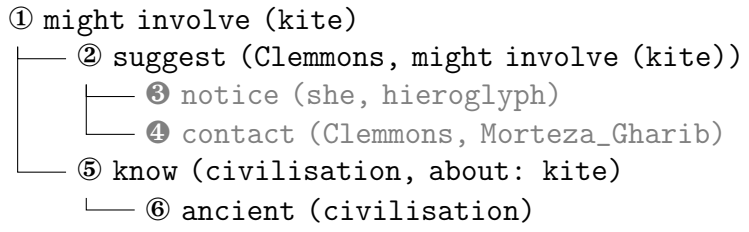
Algorithm 3 Algorithm of adding propositions within a memory cycle. The function PROPOSEATTACHMENT will be defined in Algorithm 4 (page 56). The function EXECUTEATTACHMENT is defined in Algorithm 2.

Require: *pendingPropositions*, a set of propositions that are not in *tree*
Require: *tree*, a memory tree
Ensure: all attachable propositions in *pendingPropositions* are added to *tree* as nodes, and are removed from *pendingPropositions*

```
 1: procedure AddPropositions(pendingPropositions, tree)
 2:     plans ← empty map from proposition to attachment plan
 3:     for all p ∈ pendingPropositions do
 4:         plans[p] ← ProposeAttachment(p, tree, tree)
 5:     end for
 6:     while plans is not empty do
 7:         q ← argmax_q plans[q].score
 8:         if plans[q] is an empty plan then
 9:             break
10:         end if
11:         remove q from plans and pendingPropositions
12:         subtree ← ExecuteAttachment(plans[q], tree)
13:         for all p ∈ pendingPropositions do
14:             if plans[p].site ∈ subtree then
15:                 plans[p] ← ProposeAttachment(p, tree, tree)
16:             else
17:                 newPlan ← ProposeAttachment(p, subtree, tree)
18:                 if newPlan.score > plans[p].score then
19:                     plans[p] ← newPlan
20:                 end if
21:             end if
22:         end for
23:     end while
24: end procedure
```

a node that has been added or reactivated as a result of the attachment. This information allows efficient update of the attachment plans of the other pending propositions. Specifically, how the attachment plan of a pending proposition should be updated depends on whether the planned attachment site is inside the affected subtree (Lines 14–21, Algorithm 3):

- If the proposed attachment site is inside the affected subtree, the score of this attachment plan may have been invalidated by previous tree changes. Therefore, a new attachment plan should be created by scanning the entire tree (Line 15).



Pending proposition	Site	Score
❼ could lift (kite, weight)	①	1.00
❽ huge (weight)	–	0.00
❾ realise (Gharib, could lift (kite, weight))	❹	0.02

Figure 3.5: The memory tree (upper) and the attachment plans for the pending propositions (lower) after initialization (before the first iteration). The proposition numbers are arbitrarily chosen as shorthand. Activated nodes are indicated with white circled numbers, deactivated nodes with grey circled numbers, and pending propositions with black circled numbers.

- If the proposed attachment site is outside the affected subtree, the score of this attachment plan is still valid, a site better than the current one can only be possibly found within the affected subtree (Line 17).

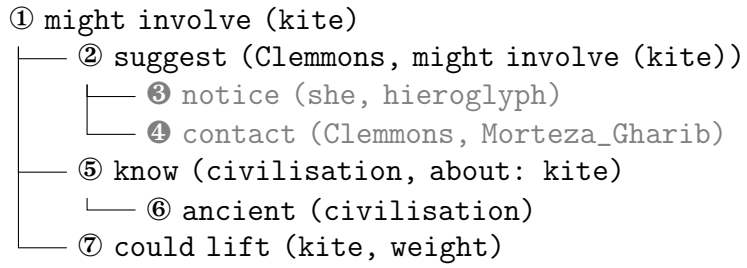
To illustrate how this iterative algorithm works, let us consider a (simplified) real situation: Figure 3.5 shows a memory tree consisting of six propositions, and three new propositions are to be attached to it. The figure reflects the status just before Line 6 (Algorithm 3), when the attachment plan of each new propositions has been set for the first time. The following are attachment plans for each of the propositions:

- Proposition 7 is to be attached under proposition 1. This is due to the overlap of the argument `kite` (proposition 5 also has this argument, but it is at a less favourable position of the tree).

- Proposition 9 is to be attached under proposition 4, due to the co-referring arguments `Gharib` and `Morteza_Gharib`. If this attachment is to be carried out, it requires reactivation of proposition 4, which is forgotten at this point.

- Proposition 8 has nowhere to attach to and is assigned with an "empty plan".

Entering Line 6 (Algorithm 3), the proposition that will be attached in the first iteration of the while-loop is proposition 7, because its attachment plan



Pending proposition	Site	Score
❽ huge (weight)	⑦	0.60
❾ realise (Gharib, could lift (kite, weight))	⑦	0.60

Figure 3.6: The memory tree (upper) and the attachment plans for the pending propositions (lower) after the first iteration. See the caption of Figure 3.5 for legend.

has the highest score. At Line 12, this attachment is executed, resulting in the memory tree as shown in Figure 3.6. The variable *subtree* now holds a pointer to the subtree that is affected by this attachment; in this case, it is the subtree rooted at proposition 7 itself. Before the second iteration, we have to update the attachment plans of the two remaining propositions, because the attachment of proposition 7 opens up new possible ways of attaching these two propositions, which have to be compared against the existing plans. For both propositions, the existing plans indicate an attachment site that is outside *subtree* Therefore Line 17 is used instead of Line 15, i.e. we try to find a better attachment site within *subtree*. The updated attachment plans are shown in Figure 3.6 as well. Note that for proposition 9, the new plan of attaching under 7 has replaced the old plan of attaching under 4, because it has a higher score. In the subsequent iterations, no updates of attachment plans occur, and propositions 8 and 9 are attached under 7 as planned in Figure 3.6.

I will now turn to the question of how an attachment plan is proposed. As a preparation, I will first describe how the score of an attachment plan is calculated in the following sub-subsection.

3.3.1.1 Evaluating an attachment plan

To understand how an attachment plan is proposed, it is necessary to first discuss the reinstatement of forgotten propositions. KvD's reinstatement strategy is not applicable to my model for two reasons. First, KvD trigger the search

for forgotten propositions only when there is absolutely no connection between any activated proposition and any new proposition. Because the strength of argument overlap is a continuous value in my model, such a scenario is unlikely to happen. Instead, we often need to choose between a weaker overlap without requiring reinstatement, and a stronger overlap with reinstatement. Second, KvD reinstate propositions only by tracing embedded propositions (cf. page 39). This is presumably because KvD regard proposition embedding as the strongest form of overlap, as the understanding of a proposition depends on the activation of its arguments. However, there is often not enough information to create embedded propositions automatically, except for embedded propositions which can be derived from a syntactic parse or coreference resolution. As a result, there are often fewer embedded propositions than necessary. Hence, I decide to use ordinary argument overlap to guide the search of propositions to reinstate, in addition to proposition embedding.

I therefore adapt KvD's strategy of proposition attachment and reinstatement so that it can work with relative argument overlap. For each node s of the memory tree, whether activated or not, I evaluate the option of attaching the new proposition p under s using the following equation. Among different options for p, the option that has the highest score becomes the attachment plan for p.

$$score(p, s) = overlap(p, s) \cdot \alpha^l \cdot \beta^n \tag{3.7}$$

In this equation, $overlap(p, s)$ refers to the proposition overlap between p and s, which will be explained in Sub-subsection 3.3.1.4. Both α and β are model parameters in the range of $(0, 1]$. The value l is the tree level (depth) of s; therefore α^l is the penalty of attaching the proposition lower in the tree. The value n is the number of deactivated nodes we need to recall in order to ensure the connectivity of the activated subgraph; hence β^n is the penalty of proposition reinstatement. Please note that proposition reinstatement is a costly memory operation that is only used as a last resort (in situation where the memory selection in the previous cycles has failed to predict and retain propositions that are useful for future connections). Therefore, β should be much smaller than α, in order to ensure that we do not use the last resort routinely.

In my experiments, I set $\alpha = 0.6$ and $\beta = 0.05$. The reasonable values of α and β depend on other parts of the system, most notably the argument overlap

β \ α	0.4	0.6	0.8
0.01	0.436	0.437	0.417
0.05	0.436	0.437	0.417
0.10	0.436	0.437	0.417

Table 3.2: ROUGE-1 (an evaluation metric which will be defined in Chapter 5) scores of the summarizer under different α and β values

module: given that the similarity of two words is a real number in the range of $(0, 1]$, the value of α should be comparable to the similarity of two related but not synonymous words. This is to ensure comparability between two situations when a new proposition containing an argument A is to be attached, and in the memory tree, there is a proposition p containing an argument B:

1. Proposition p has a child node q, which is a proposition containing both B and A. If we attach the new proposition under q, this means understanding of the new proposition is achieved by the explicit proposition q.

2. According to the argument overlap module, similarity between A and B is x. If we attach the new proposition under p, this means understanding is achieved by an implicit proposition that also connects A and B. This implicit proposition is thought to be supplied by the world knowledge modelled by the argument overlap module.

If $x < \alpha$, the first option is preferred; otherwise, the second option is preferred. To explore the influence of these parameters, I conducted an experiment using a small development corpus of two news texts and 8 human-written 100-word summaries (4 summaries per text). Because of the dependence between the parameters and the argument overlap module, the argument overlap module has to be fixed in this experiment – the lexical chain system, as will be described in Section 4.3, is used with its default setup. The performance of my summarizer (Table 3.2) is highest when $\alpha = 0.6$, whereas β has no effect (at least not visible on this dataset). If a large dataset of text and summaries is available, it is possible to use the same kind of experiment to estimate the optimal values of α and β, but which are specific to that text genre and the argument overlap module employed by that experiment setup.

Let us consider an example of scoring of attachment options. In Figure 3.7,

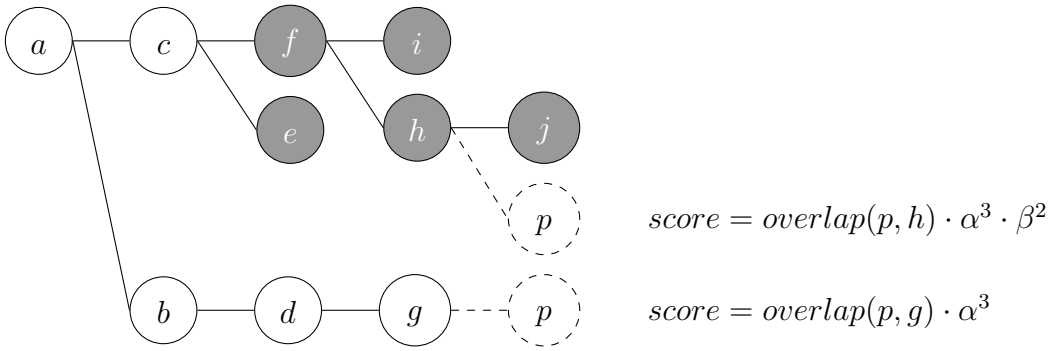


Figure 3.7: Two possible attachment sites for a new proposition

white nodes represent activated propositions; grey nodes represent forgotten propositions. There are two possible attachment sites h and g for the new proposition p.[1] If p is attached under h, the score of attachment would be $overlap(p, h) \cdot \alpha^3 \cdot \beta^2$, because h is situated at level 3, and two propositions f and h need to be recalled so that the path from a to p would consist only of activated nodes. The score of attaching p under g would be $overlap(p, g) \cdot \alpha^3$, because g is on the same level as h, but no propositions need to be reinstated. Thus, whether reinstatement would occur in this example depends on whether $overlap(p, h) \cdot \beta^2$ or $overlap(p, g)$ is greater.

In this example, I have assumed that reinstated propositions stay in their original locations; only their memory status is changed from deactivated to activated. But it is also possible to move a node (which is equivalent to transplanting the subtree rooted at that node) in order to reduce the number of required reinstatements. I will now explain how this is done.

3.3.1.2 Reducing reinstatements

Because proposition reinstatement is costly, it is desirable to reduce the number of propositions to reactivate by changing the memory tree beforehand in preparation of the reinstatement. In the previous example, if we move the subtree rooted at h to attach under b, assuming non-zero $overlap(h, b)$, we would not need to reactivate f, and the attachment site for p would be one level higher than before (Figure 3.8). This change increases the score of the option of attaching p under h, from $overlap(p, h) \cdot \alpha^3 \cdot \beta^2$ to $overlap(p, h) \cdot \alpha^2 \cdot \beta$.

[1] In a real situation, every node in the memory tree should considered, but in the example we only compare these two options.

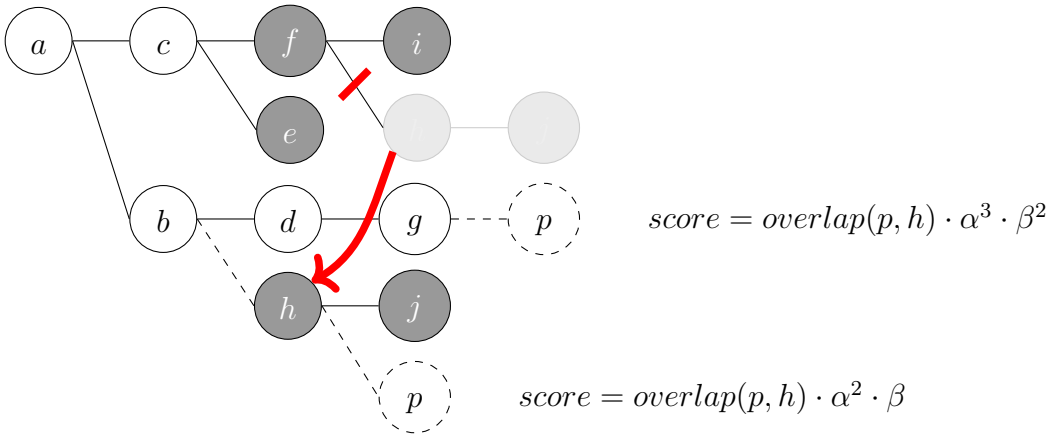


Figure 3.8: Transplanting a subtree to reduce the cost of reinstatement

In theory, there are a large number of possible changes that can be possibly performed. It is even possible to find the optimal tree that would maximize the score of attaching the new proposition to a particular node. However, I assume that the memory tree represents a stable, established interpretation of the text. Any dramatic change to the memory tree is presumed to be accompanied by a huge cognitive cost.

For this reason, as well as for the sake of efficient computation, I only consider the option of moving one node. Two further restrictions are: (1) the node to be moved has to be in the path from the root to the attachment site in question, and (2) the new parent (i.e. re-attachment site) of the moved node has to be activated. In Figure 3.8, the only nodes to be considered for moving are h and its ancestors.

So far I have described the methods of evaluating and improving options of attachment. I will now proceed to the algorithm of finding the attachment plan for a proposition.

3.3.1.3 Proposing an attachment plan

My algorithm of determining the attachment plan of a proposition `p` is presented in Algorithm 4. Its overall structure is as follows: It evaluates every node within `searchSpace` as a potential site of attachment for `p` (Line 3), and returns the highest scoring attachment plan for `p`. The parameter `searchSpace` could be the entire memory tree, or a subtree of it, depending on how this function is called by AddPropositions (cf. Algorithm 3).

When evaluating a particular node `site`, the ancestor nodes of `site` are

Algorithm 4 The algorithm to propose the attachment plan for a proposition. The constructor AttachmentPlan creates and returns an instance of the structure AttachmentPlan (cf. Algorithm 2). The *score* of an AttachmentPlan is computed automatically before it is used according to Equation 3.7; the empty plan is a special instance that has a lower score than any real attachment plans. The iterators Traverse and TraverseActivatedNodes yield (for the latter, activated) nodes of a tree in breadth-first order, and in descending order of recency for nodes of the same depth. The criterion of a permissible transplant is described on page 57.

Require: p, a proposition not in *tree*
Require: *searchSpace*, a memory tree (equals *tree* or a subtree of it)
Require: *tree*, a memory tree
Ensure: an attachment plan is computed for p to attach as a child of a node in *searchSpace*; the memory tree itself is not changed

```
 1: function ProposeAttachment(p, searchSpace, tree)
 2:     bestPlan ← empty plan
 3:     for all site ∈ Traverse(searchSpace) do
 4:         plan ← AttachmentPlan(node = p, site = site)
 5:         plan.transplant ← empty plan
 6:         for all a ∈ the path from site to tree.root do
 7:             if a = tree.root ∨ a.parent is activated then
 8:                 break
 9:             end if
10:             for all s ∈ TraverseActivatedNodes(tree) do
11:                 t ← AttachmentPlan(node = a, site = s)
12:                 if t is permissible ∧ t.score > plan.transplant.score then
13:                     plan.transplant ← t
14:                 end if
15:             end for
16:             if plan.transplant ≠ empty plan then
17:                 break
18:             end if
19:         end for
20:         if plan.score > bestPlan.score then
21:             bestPlan ← plan
22:         end if
23:     end for
24:     return bestPlan
25: end function
```

considered for transplantation in bottom-to-top order (Line 6), starting from the parent node of `site`. This process terminates when it is no longer possible to find a better node to move, i.e. moving any other ancestor would not result in a larger reduction in the number of reinstatements. The termination condition is spelled out as satisfying either of the following conditions:

1. The parent of the ancestor node (let us call it `a`) is activated (Line 8). Because activated nodes are guaranteed to be connected, the path from an activated ancestor to the root does not contain any deactivated nodes, which are the target of reduction.

2. A transplant plan that moves `a` is found (Line 17). In this case, moving a higher-level node would only result in a smaller reduction.

Let us zoom into the question of where the ancestor node `a` should move to (Lines 10–15). I test all possible activated nodes as a possible attachment site for moving the subtree rooted at `a` to, and choose the best among these. A transplant plan is considered permissible if it satisfies both of the following conditions (let us denote the current parent of `a` as `x`):

1. $score(\texttt{a}, \texttt{s}) \geq score(\texttt{a}, \texttt{x})$, i.e. `s` is not a worse attachment site for `a` than `x`. Although `x` was the highest scoring attachment site when `a` was new (assuming that `a` has not been moved since its first attachment), this may no longer hold because of changes of memory status and location, or because `s` became available only after `a` was added.

2. $score_{after}(\texttt{p}, \texttt{site}) > score_{before}(\texttt{p}, \texttt{site})$, i.e. the score of attaching `p` under `site` would increase if the move is performed. This condition ensures that the transplant is advantageous for the new proposition. It is often met automatically because a transplant reduces the number of propositions to reinstate.

In the entire algorithm of adding propositions, there is only one thing that has not been specified, namely the overlap between two propositions. I will now explain how this is calculated.

3.3.1.4 Proposition overlap

The overlap of the content of two propositions p and s, where p will be attached under s, has been denoted as $overlap(p, s)$ in this chapter. The computation of argument overlap is the topic of the next chapter. The proposition overlap can be regarded as the sum of individual argument overlaps between each argument of one proposition and each argument of the other proposition.

Instead of a simple summation, there are two issues to consider when computing proposition overlap. The first one concerns the special status of functors. Because functors are mostly relational concepts or verbs, the dependency of their meaning on their arguments is normally higher than the dependency of the meaning of arguments on their functors. While KvD consider an argument to be a concept, a functor is only considered as a part of its proposition, and overlap using functors is not observed in the *Bumper Stickers and the Cops* example. Overlap using embedded propositions takes over the role of functor overlaps.

In my simulation, the situation is slightly different, as there are fewer embedded propositions. To recover the loss of overlap, I allow an argument of the child proposition p to overlap with either the parent proposition s itself (i.e. the argument is an embedded proposition) or the functor of s. For example, let us consider the following two propositions:

$$s = \texttt{assassinate (politician)}$$
$$p = \texttt{condemn (assassination)}$$

If the argument `assassination` is replaced by s by event coreference resolution, attaching p under s would be obviously possible. If this does not happen, such an attachment is still possible due to the lexical overlap between the argument of p and the functor of s. Note however, that the functor of the child proposition does not participate in the overlap detection, and hence proposition overlap is not always symmetric (in contrast, argument overlap is).

The second issue concerns conflicting interpretations of arguments. A consequence of using continuous argument overlap is that an argument may have many non-zero overlaps. Figure 3.9 illustrates this by displaying all argument overlaps between two example propositions s and p as edges of different weights (representing different degrees of overlap, in this example computed using the

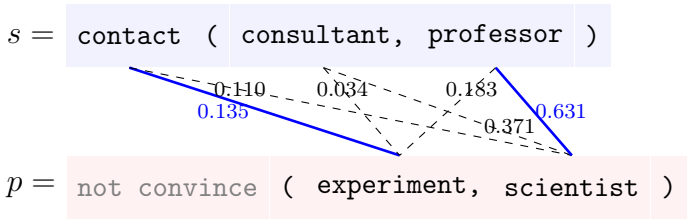


Figure 3.9: Computation of proposition overlap from argument overlap

GloVe vectors to be detailed in Section 4.2). Supposing we would like to compute $overlap(p, s)$, the functor of p cannot participate in the overlaps. Notice that all participating words have more than edges; most problematically, one argument of p, namely `scientist`, has relatively high overlap with both arguments of s, `consultant` and `professor`. I assume arguments in the same proposition represent different concepts, and therefore an argument of p can have at most one corresponding argument in s, and vice versa. I select the interpretation that maximizes the total strength of remaining overlaps, which is denoted as $overlap(p, s)$. This interpretation is defined as the maximum weight matching of a bipartite graph in which edges exist between two disjoint sets of nodes representing arguments of the two propositions in question. In this example, the remaining edges are drawn as solid blue lines, and $overlap(p, s) = 0.631 + 0.135$.

3.3.1.5 Computational complexity

Empirically, a memory cycle has a reasonable running time on average. The computational complexity of my proposition attachment algorithm can be analysed with or without the two following constraints: (1) the number of possible reinstatements in a memory cycle, and (2) the number of new propositions in each sentence. I will start with an analysis without these constraints.

Let us suppose there are currently n propositions in the memory tree, among which the number of activated propositions is a small constant, and assume constant running time of the computation of proposition overlap. The complexity of proposing an attachment plan by searching the entire memory tree in the way of ProposeAttachment (Algorithm 4) is $O(n^2)$, because there are n nodes to search, a tree node can have at most $n-1$ ancestors in the worst case, and testing an ancestor for transplant takes constant time due to the constant number of activated propositions.

Supposing there are m pending propositions to be attached in a memory cycle, the time complexity of a memory cycle is $O(m^2(n+m)^3)$. I arrived at this result by breaking a memory cycle into two parts:

1. Computing the initial attachment plans for m propositions takes $O(mn^2)$ time, i.e. $O(n^2)$ per proposition as we have just seen above.

2. The running time of updating the attachment plan for one proposition is bounded by $O((n+m)^3)$, following a similar analysis as before, except that there are at most $n+m-1$ nodes in the tree. It is cubic instead of quadratic, because all nodes of the tree could be activated due to attachments and reinstatements. As there are at most m attachments in a memory cycle, and attachment plans of at most $m-1$ pending propositions are updated after every attachment, the complexity of the attachments is $O(m^2(n+m)^3)$.

In order to obtain a tighter estimate of the average running time, two assumptions can be made:

1. The number of reinstated propositions in a memory cycle can be assumed to be a constant. Reinstatement of proposition is rare by design, and as a result, the number of activated propositions never grows to the order of the total number of propositions in the memory tree. Because a subtree can be transplanted only by attaching under an activated node, the complexity of a memory cycle is $O(m^2(n+m)^2)$ under this assumption.

2. m can be assumed to be a constant. In a memory cycle, the propositions pending attachment include both the propositions of the current sentence, and propositions from previous sentences which failed to attach in the previous memory cycles. A proposition fails to attach only if it does not have even the smallest amount of overlap with any existing proposition, which is unlikely in a coherent discourse and using a continuous notion of overlap. On the other hand, the number of propositions from the current sentence is bound by the normal use of natural language, or alternatively by a limit set manually (forcing longer sentences to be processed in multiple cycles).

Under these two assumptions, the time complexity of a memory cycle is $O(n^2)$. Note that no assumption about the tree shape is made.

In order to improve the runtime, I also perform a combination of optimizations, which have been omitted from the pseudocode above for clarity. These optimizations concern tree pruning and memoization. Here are a few examples of such optimizations:

1. On Line 3 of Algorithm 4, a `site` that has zero overlap with `p` is immediately skipped, without trying to improve the attachment score.

2. Proposition overlap is memoized across cycles. Therefore, in cycle k, if a new proposition fails to attach, in cycle $k+1$ we only need to test whether it can attach under a proposition that became available in cycle $k+1$.

3. The tree data structure itself also helps me impose the constraints in an efficient way. The search for a better attachment plan (Line 3 of Algorithm 4) is abandoned if all attachment sites to be encountered in the future are guaranteed to be no better than the current best site . This case can be easily detected because an upper bound of proposition overlap can be determined, and because the memory tree is traversed level by level.

4. The search for a place to attach a transplanted subtree (Line 10 of Algorithm 4) is terminated if a memory location that cannot possibly satisfy either of the two conditions of a permissible transplant is reached. For example, given the model parameters $\alpha = 0.6$ and $\beta = 0.05$, transplanting a subtree to a location that is six levels deeper than before in order to reduce one reinstatement is not worth considering, because the fact that $\alpha^6 < \beta$ makes the second condition impossible to meet.

3.3.2 Controlling the tree shape

In the previous section, I have described how propositions are added to the memory tree, which is the most important part of my model. Two questions still remain, and I will now address them: (1) How is the root of the memory tree determined in the first cycle, and how does the root change in the subsequent cycles, if at all? (2) How to select the propositions to remain activated at the end of each cycle?

3.3.2.1 Determining the root

During the first memory cycle, the root of the memory tree needs to be selected among the propositions of the first sentence. I use a directed graph G of propositions as an auxiliary data structure to determine the root. In this graph, the distance of an edge from proposition p to q is defined as the reciprocal of their proposition overlap, i.e. $\frac{1}{overlap(q,p)}$. The node which has the highest closeness centrality in G is then added to the memory tree as the root, and is removed from the collection of pending propositions.

Closeness centrality of a node u is the reciprocal of the sum of the shortest path distances from u to all other nodes. The advantage of using closeness centrality with edge distance defined as above is that weak overlaps are automatically excluded during the search for the shortest paths. If two or more propositions are tied for the highest centrality, I consider the functors of these propositions and choose the one whose functor is closest to the main clause of the first sentence in a syntactic parse.

Because argument overlap is continuous in my model, it is unlikely that new propositions cannot be attached anywhere. Hence there is no practical need for multiple disconnected trees as described in Kintsch and Vipond (1979). However, for robustness, my system contains a backup scheme which would reset the memory tree if the content of the tree is considered useless for processing new sentences.

This scheme is triggered if pending propositions which fail to attach outnumber the size of the memory tree. Once triggered, the scheme works as follows: A directed graph G of pending propositions is created in the same way as the auxiliary data structure for determining the root. Let u denote the node that has the highest closeness centrality in G. I now test whether making u the new root would allow more propositions to be attached than the current tree has. If the number of nodes in G reachable from u is greater than the size of the memory tree, installing u as the root would satisfy this requirement. In this case, the memory tree is reset, and all nodes in it become pending propositions. I make u the root of the new memory tree, and add all pending propositions to the memory tree in the same way as usual.

3.3.2.2 Changing the root

Having a good root is important. As I have described in Sub-subsection 3.2.1.1, the root should reflect the general topic that is under current focus. Because propositions which share arguments with the root or high-level nodes will be attached higher than those which do not, if the root is representative of the document content, the tree levels mirror the generality of information well.

In the KvD model, the root of the coherence tree never changes after it has been determined in the first memory cycle. This is too limiting because the most central idea of a text is not necessary expressed in the first sentence. For example, an article could begin with background knowledge that has only a very vague connection to the actual topic, or with an anecdotal lead designed to engage the reader.

I want the root to change only if a major topic change occurs in text, but not when minor local topic variation occurs. My algorithm therefore tries to strike a balance between being reactive to the text and being conservative enough to allow the memory tree to grow in an overall stable fashion without constant reorganization.

I do this as follows: At the end of every memory cycle (Line 18 of Algorithm 1), a decision is made whether a root change should take place. I consider each child node of the current root as a candidate for the new root, and check whether the subtree rooted at this child node is larger than the rest of the tree. Let v denote the current root, and u a child of v. V is the set of all nodes, and U the set of nodes in the subtree rooted at u. If u becomes the new root, the edge from v to u is inverted. This means that the nodes in U will be promoted upwards by one tree level, but the other nodes, i.e. the members of $V \setminus U$, will move down. Intuitively, a root change is beneficial if more nodes are promoted than moved down, i.e. $|U| > |V \setminus U|$.

But the memory status of the nodes must be taken into account when making this decision, as only the subgraph of the activated nodes is visible to under normal operation (i.e. when no reinstatement takes place). Therefore, when computing the size of subtrees, forgotten nodes should be assigned smaller weights than activated nodes. In my experiments, the weight w_p is 1 if p is activated, 0.05 if it is not (the choice of value is consistent with the discounting factor of forgotten propositions β, defined on page 52). The condition for u becoming the new root

is:

$$overlap(v,u)\sum_{p\in U} w_p > overlap(u,v)\sum_{p\in V\setminus U} w_p \tag{3.8}$$

Proposition overlap is involved because it is not always symmetric, as I have explained in the Sub-subsection 3.3.1.4. If inverting an edge would render the proposition overlap weaker, the subtree U must win by a larger margin in order to initiate root change.

The sensitivity of the root to new information in the recent cycles is controlled by how the weight w_p is defined. If w_p for forgotten propositions is small, root change would be more frequent. For processing long texts, because the number of forgotten propositions grows over time, it may be necessary to gradually decrease w_p for forgotten propositions, so that the total weight of forgotten propositions is bounded in proportion with the number of activated propositions.

If a root change happens, the same detection mechanism is applied again on the resulting memory tree. The procedure is repeated until no root change is possible anywhere. For example, it is possible that a grandchild of the current root rises to root status in two iterations. In pseudocode (Algorithm 1), the detection mechanism is represented by the function AdjustRoot. In order to decide whether the root change detection should be repeated, this function returns a boolean value indicating whether a root change has occurred. In my actual implementation, I maintain the size of every subtree as an attribute of the root of the subtree, so that the sum of weights can be computed efficiently.

3.3.2.3 Leading-edge strategy

My algorithm of selecting propositions to remain activated (MemorySelect in Algorithm 1) is identical to KvD's leading edge strategy. In my experiments, the memory size is 5, or 40% of the number of activated propositions before selection, whichever is larger. This is influenced by KvD's choice of 4, as well as Miller's Law (Miller, 1956) in psychology, which states that the number of information chunks that can be held in human working memory is typically 7 ± 2. The use of the alternative limit by proportion allows us to accommodate a long sentence in one memory cycle (by retaining more propositions than allowed by the fixed memory size), without having to devise a way of splitting the content of a sentence into multiple cycles.

Max size or Max proportion	4 30%	5 40%	6 50%	7 60%	8 70%	9 80%
ROUGE-1 score	0.435	0.437	0.430	0.395	0.402	0.381

Table 3.3: ROUGE-1 (an evaluation metric which will be defined in Chapter 5) scores of the summarizer under different memory sizes

I have also explored the effect of different memory sizes using the same data as used in Sub-subsection 3.3.1.1. The experiment confirms that my choice of memory size results in better performance than the other values tried (Table 3.3). Whether the optimal memory size depends on the number of propositions per sentence and the summary:source ratio (the ratio of summary length to source text length) remains an open question. In this experiment, there are on average 7.6 propositions per sentence, and the average summary:source ratio is 10%.

The leading edge strategy depends on an attribute of proposition called recency. As we have seen in Sub-subsection 3.2.1.3, KvD decide the recency of a proposition not only by its proposition number (i.e. the textual position of its functor), but also by the time when it is added or reinstated to the coherence graph. Therefore, I remember for each proposition the time slot when the proposition was attached to its current site (either in initial processing, or in reactivation is such reactivation happened). When comparing propositions by recency, the time slot is compared first, and if there is a tie, the proposition number is then compared.

As with the KvD model, the summary worthiness of a proposition is determined by the number of memory cycles it has remained activated (henceforth called the "memory count"). According to KvD, human readers recall propositions probabilistically. In contrast, my summarizer outputs propositions deterministically in descending order of their memory counts. There are different ways of realizing the summary propositions into natural language, which I will explain in Chapter 5.

3.4 Full connectivity baseline

In the last section, I have presented a way to simulate comprehension, which is modelling the process of comprehension as a series of memory updates (addition, forgetting, and other changes). The basis of this approach is the assumption that working memory is limited in capacity, hence a plausible simulation has to involve a dimension of time, as opposed to processing an entire document at once. The

same idea also underlies some practices of machine learning, including recurrent neural networks and reinforcement learning. However, it is still necessary to test whether the incremental processing model is advantageous when compared to a global optimization for this particular task.

In order to test whether the incremental memory cycles contributes to summarization performance, I have created a competitor system, denoted as the Graph summarizer. This system has access to the same information as my summarizer, i.e. it uses the same propositions and computes argument overlap between propositions in the same way. The only difference is that, instead of a complete simulation, it models text comprehension in the simplest form – representing an entire document as a fully-connected graph of propositions. The importance of a proposition can be determined by applying different kinds of centrality algorithms (Newman, 2010) to the graph. The intuition is that well-connected propositions and propositions that are referred to by other propositions should be ranked highly, similar to their status in memory cycles. Because an edge can exist between any pair of propositions with non-zero argument overlap, such a graph can contain more information than a memory tree, making the Graph summarizer a potentially strong competitor.

In the experiments (Chapter 5), I will compare the performance of my summarizer to five variants of the Graph summarizer using the following three graph centrality algorithms:

Betweenness centrality: I create an undirected graph, where an edge exists between any two nodes (propositions) which have non-zero proposition overlap. The distance of an edge between node u and node v is defined as the reciprocal of the maximum of $overlap(u, v)$ and $overlap(v, u)$ (recall that proposition overlap is directional).

The betweenness centrality of a node u is defined as $\sum_{s \neq u \neq t} \frac{\sigma_{st}(u)}{\sigma_{st}}$, where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(u)$ is the number of those paths that pass through u. If a node is involved in many shortest paths connecting other nodes, it is regarded as important. For example, Figure 3.10 is a graph on which betweenness centrality is computed. The nodes which have the highest betweenness centrality are often situated between a cluster of closely connected propositions and the rest of the graph, serving as gateways to subtopics of the document. This

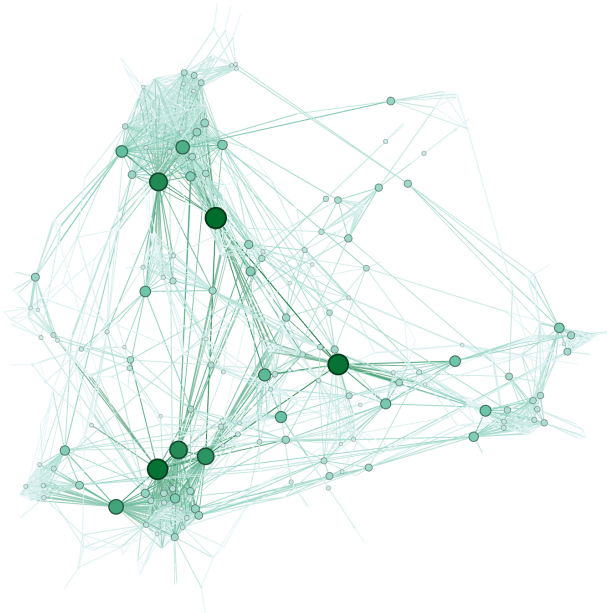


Figure 3.10: A Gephi representation (using the Force Atlas layout) of the graph for document *In search of the holy grail* from my corpus. Bigger nodes have higher betweenness centrality.

variant of the Graph summarizer is called Graph-B in Chapter 5.

Closeness centrality: I use this type of centrality on both a directed graph and an undirected graph. The undirected graph is created in the same way as in betweenness centrality. The directed graph is created by defining the distance of an edge from node *parent* to node *child* as $1/overlap(child, parent)$ (the names *parent* and *child* reflect their status if they were put in the memory tree).

The closeness centrality of a node u is defined as $\frac{n-1}{\sum_{v \neq u} d(u,v)}$, where $d(u, v)$ is the shortest-path distance from u to v, and n is the number of nodes reachable from u (a node that is not connected to any other node is assigned with zero centrality). A node is regarded as more important if it is on average closer (in terms of shortest paths) to other nodes. The systems which use directed and undirected graphs are called Graph-CD and Graph-CU, respectively.

Eigenvector centrality: The eigenvector centrality of a node is a relative score based on the eigenvector centrality of nodes that are connected to it, weighted by the weights of the incoming edges. It is used in PageRank as well as several graph-based summarizers (cf. Subsection 2.1.3). Mathematically, finding

the scores amounts to finding the eigenvector for the largest eigenvalue of the adjacency matrix of the graph, hence the name.

This type of centrality is also applicable to both directed and undirected graphs. In a directed graph, the weight of an edge from *child* to *parent* is $overlap(child, parent)$. Note that the directionality of the edges is the opposite of that of the edges in Graph-CD. In eigenvector centrality, we rank the importance of a node by the importance of the nodes which would be descendants of this node in the memory tree. Therefore, the edges should be in the same direction as this influence flows. In an undirected graph, edge weight is the maximum of proposition overlap in both directions. The systems which use directed and undirected graphs are called Graph-ED and Graph-EU, respectively.

Chapter 4

Argument overlap

Argument overlap models the local coherence of a text. The definition of argument overlap is conceptually simple, namely two arguments symbolizing the same "concept", which may, for example, refer to an entity, an event, or a class of things. However, modelling argument overlap is complex, as it is related to a broad range of linguistic phenomena, including coreference, anaphora, paraphrasing, and textual entailment. In this chapter, I will present my exploration of methods to model argument overlap.

Research of argument overlap has not only theoretical value for document understanding, but also practical importance for my summarizer. I have described my algorithm of simulating reading comprehension in the previous chapter. Given a method of detecting argument overlap, the algorithm itself is deterministic and is only affected by a small number of parameters such as the memory size. Therefore, the quality of a simulation largely depends on the quality of argument overlaps involved in that simulation.

Coreference resolution, the NLP task of detecting linguistic expressions that refer to the same entity, is a natural choice for argument overlap detection. In psycholinguistic tradition, local coherence is based on entities, and many theories are centred around the life cycle (introduction or reference) and the salience (cognitive accessibility) of entities (Halliday and Hasan, 1976; Prince, 1981; Gundel et al., 1993; Strube and Hahn, 1999). Computationally, Barzilay and Lapata (2005) measure local coherence by representing a discourse as a two-dimensional grid, whose rows correspond to sentences and columns to entities.

However, coreference resolution (in its original sense) is not a perfect match

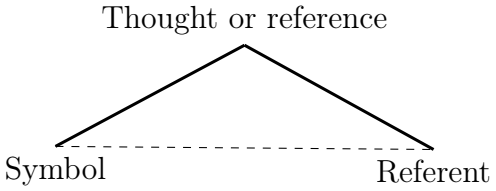


Figure 4.1: The triangle of meaning

for argument overlap. Let us consider the difference between coreference and argument overlap with respect to the triangle of meaning (Ogden and Richards, 1923, Figure 4.1). In this triangle, a linguistic symbol can symbolize a thought or reference, and a reference can refer to a referent. However, there is no direct relation between a symbol and a referent. Argument overlap is the identity of thought or reference, which I call "concept". Whereas what coreference resolution tries to achieve is to establish an equivalence class of symbols which stand for the same referent, which, according to the triangle of meaning, has to be done by modelling the concept in the middle. Therefore, I consider argument overlap to be more fundamental to text understanding than coreference.

KvD explain the coreference phenomenon in a way that is consistent with the analysis above. For example, let us consider Frege's (1948) example of "*the morning star is the evening star*". First, linguistic symbols are assimilated into concepts. Here, MORNING STAR and EVENING STAR are two different concepts, even though we know they refer to the same referent.[1] Second, the knowledge about different concepts sharing a referent is represented as a proposition. A proposition involving the concept MORNING STAR and a proposition involving the concept EVENING STAR can only connect to each other via the proxy of an intermediate proposition such as BE (MORNING STAR, EVENING STAR). Such an intermediate proposition may be supplied by the text itself, or by the reader's own knowledge.

An important exception to the rule of using different concepts is anaphoric coreference, or what KvD call "implicit coreference". Anaphora is the use of an expression which depends on another expression (which is called the antecedent) for its interpretation. For example, in the case of using the pronoun "*he*" in the sentence "*He is 35*" to refer to CLARK KENT, the proposition corresponding to

[1] In contrast, KvD consider "*police*" and "*law enforcement officers*" as different linguistic symbols for the same concept.

this sentence would contain the concept CLARK KENT, instead of a hypothetical HE or MAN. I attribute this treatment to the fact that the pronoun "*he*" depends on its context for its interpretation, and is therefore not an independent concept. Instead of a pronoun, using the noun phrase "*the president*" to refer to a president in context is also an instance of anaphoric coreference.

I do not plan to discuss the full complexity of coreference here, but it is already clear that coreference can be regarded as a result inferred from the meaning of a text using knowledge both internal and external to the text. An explanatory model of coreference should separate the decoding of symbols into concepts from the application of knowledge (represented as propositions involving those concepts). However, in the world of NLP, coreference resolution is studied as a unitary task. This of course has practical reasons, but renders it insufficient for argument overlap. In Section 4.1, I will explain what useful information we can gain from coreference resolution, and more importantly, identify the side effects of basing coreference on a simplistic view of semantics.

Considering the NLP technologies that exist today, I have devised a two-stage method to detect the overlap of two arguments, which are words or phrases from two propositions. In the first stage, an argument which is in an anaphoric coreference relation is replaced by its antecedent. This is done by using a coreference resolution system, because most coreference relations detected by these systems are anaphoric. If the two arguments are already identical after the first stage, the overlap between them is 1, i.e. an 100% overlap. Otherwise, a second stage that accounts for the meaning of the arguments is performed, which defines overlap as a real number between 0 and 1. Intuitively, the overlap value represents the probability that the two expressions convey concepts that can be regarded as the same in that particular discourse. How this value should be computed, however, is worth further discussion.

The second stage is required because there are many cases of argument overlap which do not involve actual referents, or in which the referents are difficult to determine. The overlap of abstract concepts such as JUSTICE is an obvious example in which overlap is between thoughts rather than references. The reason why the strength of overlap determined in the second stage has to be continuous can be understood from two perspectives: In one perspective, there are very few cases in which the meanings of two different expressions are absolutely identical, but what is relatively common is that the distinction in meaning is so insubstantial

in that context that they can be regarded as expressing approximately the same concept. For example, strictly speaking, "*cigarette manufacturers*" and "*the tobacco industry*" represent different concepts, but they can be conflated in a report about smokers' health.[2] Pairs of expressions that are more similar in meaning compared to other pairs are more likely to be interpreted as representing the same concepts. In the other perspective, two different but related concepts should amount to an indirect overlap with the help of common-sense knowledge. For example, CAR and VEHICLE are different concepts, but a reader would have no trouble understanding a sentence about CAR within a paragraph about VEHICLE, because coherence is maintained via the hypernymy relation between the two concepts. To simulate this, which human readers naturally perform based on their semantic memory, we need not only the semantic relations between words, but also the relative strength of these associations. The stronger the association, the more likely the indirect overlap is used by the comprehension process.

Both perspectives in the previous paragraph prompt for a relative and continuous measure of overlap. However, the qualities they depend on are different, i.e. similarity vs. relatedness. The question of defining similarity or relatedness is non-trivial by itself, and I will use a pair of more formal notions when I compare computational methods for word similarity later in this chapter. To evaluate lexical similarity models, computational linguists have created datasets consisting of word pairs rated by human annotators according to various criteria of similarity, such as WordSimilarity-353 (Finkelstein et al., 2001) and SimLex-999 (Hill et al., 2016). It is not immediately clear which definition and which method are the most appropriate for simulating the argument overlaps used in text comprehension.

The approach I take is an empirical one. To model word meaning, I have experimented with two types of systems: vector space models of word meaning and the lexical chain method. How I implement and use these methods will be detailed in Section 4.2 and 4.3, respectively. Both types of systems are used in conjunction with a coreference resolver. I choose to use the lexical chain in my final system, because it results in better summaries (which reflect better simulation of comprehension). The experiments and the evaluation of these systems are in

[2]This phenomenon concerns the granularity in the semantic space: two concepts are so similar that it is not necessary to distinguish them in that context. It is different from metonymy, in which one concept stands for another concept that is closely related to it. Metonymy falls under the discussion of the other perspective.

Chapter 5.

A major challenge for argument overlap is the treatment of compositionality in language. Compositionality (apart from quantification) is not treated specially in KvD-style propositions, which are products of a logic-based view. For instance, "*young person*" would be represented as the proposition YOUNG (PERSON), but when the phrase is used in another proposition as an argument, instead of embedding the proposition, only the concept PERSON is used, with the understanding that this PERSON is a merely symbol for the concept which can be equally written as YOUNGSTER or x_1.[3] More complex modifications such as relative clauses are similarly treated. In this way, the meaning of a concept (as a variable) is not static, but is influenced by the propositions which it participates in. Hence, from a symbolic perspective, I can formulate the task of detecting concepts in a document as finding a set of variables $X = \{x_1, ..., x_k\}$, and a mapping of the content words $W = \{w_1, ..., w_n\}$ to X, such that both the explicit propositions in the text and any implicit propositions (originated from e.g. the meaning of words, world knowledge, and constraints for well-formed discourse) are satisfied. To illustrate this idea, let us consider a coreference resolution task of resolving the phrase "*the victim*" after the statement "*Emily was killed by the guerrillas*". The information-status of the phrase entails that it must be assigned with a previously given concept, and EMILY is selected because it is compatible with the meaning constraint of the word "*victim*" by its participation in the proposition KILL (GUERILLA, EMILY). In conclusion, compositionality would not be a problem if symbolic inference could be done robustly on natural language. In fact, it has long been known that there are instances of the coreference phenomenon which can only be resolved by inference (Winograd, 1972), which still receive research attention in modern time (Peng et al., 2015).

On the other hand, there are also sub-symbolic approaches towards compositionality. Vector representations of word meaning can be interpreted from the structuralist view as decomposing the meaning of a word into basic units, except that the vector dimensions are latent and continuous, whereas categorical basic units with clearly interpretable meaning are traditionally used. Composition therefore can be done through manipulation over these dimensions. Many modes of composition have been proposed for vector space representations, including

[3] x_1 is subject to two propositions $\text{PERSON}(x_1) \wedge \text{YOUNG}(x_1)$.

addition, addition augmented with similar words of the predicate, element-wise multiplication, tensor product, and circular convolution (Mitchell and Lapata, 2010). Currently, a weighted average of word embeddings remains a strong baseline in detecting textual similarity (Arora et al., 2017); the fact that it does not depend on word order or grammatical relations indicates room for improvement in the area of compositional distributional semantics.

From the sub-symbolic point of view, vector space representation of propositions can be obtained by composing its predicate and arguments via one of the modes of composition mentioned above. Doing so would allow us to determine whether a word and a proposition represent the same concept, or whether two propositions represent the same concept. More generally, the representation of a concept is adjusted by its participation in propositions, such that both the inherent meaning of the word used to represent it and any additional knowledge about this concept are encoded in the same vector space. Thus, a soft classification of concepts can be obtained by clustering compatible concepts. The obstacle in this path, however, is the lack of a way to represent individuals in vector space: There is not yet a methodology of creating a vector representation for a "*James*" to account for his role as a student in computer science and his falling love with "*Mary*", such that these pieces of information can be queried on the vector – although this is clearly possible using a knowledge-relational graph of symbols. The treatment of compositionality ultimately depends on a better integration of the symbolic and the sub-symbolic representations, but I am afraid that it is beyond the scope of this chapter. Instead, I will touch only the handling of non-compositional (or idiomatic) phrases in the following discussion.

4.1 Coreference resolution

Coreference resolution has been studied for decades (in some cases, only anaphoric coreference is studied), but remains a difficult NLP task today. Interestingly, early works on coreference resolution are typically based on discourse models such as Centring Theory (Grosz et al., 1983, 1995). Discourse models can predict the status of discourse referents in a reader's mind – a goal that is shared with KvD's coherence graph. According to Centring Theory, a sentence in a discourse has a set of forward-looking centres C_f and exactly one backward-looking centre C_b,

which is a also member of C_f. The claims of the theory include: (1) members of C_f are ranked by their grammatical functions in the sentence (for instance, subject is ranked higher than object); (2) C_b of a sentence is the highest-ranked element of C_f of the previous sentence; (3) C_b of a sentence is most likely (among members of C_f) to be realized as a pronoun. Anaphora can be resolved by keeping track of C_f and C_b (their updates correspond to four types of transitions) while processing a text sentence by sentence. It would be theoretically valuable if a similar methodology could be established which models focus upon the memory tree instead of these centres.

With the advance of statistical NLP, researchers have engineered a variety of features to detect pairs of co-referring expressions. For example, Soon et al. (2001) use 12 features, including distance in text, whether an expression is a pronoun, whether both strings match, whether the semantic classes of both expressions (ten classes defined by WordNet hypernyms) match, etc. Based on the pairwise results, the equivalence classes of expressions can be constructed using simple clustering algorithms (Cardie and Wagstaf, 1999; Ng and Cardie, 2002), graph partitioning (McCallum and Wellner, 2005), or integer linear programming (Denis and Baldridge, 2007; Peng et al., 2015). An advanced clustering method is able to balance both positive and negative (reflecting incompatibility) decisions, which basic methods such as taking the transitive closure of all detected connections (Haghighi and Klein, 2009) are unable to do. In regards to machine learning, there is also a choice of the learning algorithm, as well as a choice of the method to select training examples from an annotated corpus.

As improvements to the classical mention-pair model, in which classification is performed locally on pairs of expressions and clustering is done as a separate stage, different methods have been experimented to exploit the global context. One approach is to use features that are based on entire clusters instead of pairs (Luo et al., 2004; Culotta et al., 2007; Clark and Manning, 2016), which models the intuition that members of a cluster should be compatible with each other more directly. An other approach is to rank the candidate antecedents of an expression relatively to each other (Denis and Baldridge, 2008). The idea of treating the strength of connection or the salience of entities as relative to competitors applies to many approaches, including my use of argument overlap.

In my system, I use the Stanford coreference resolver (Raghunathan et al., 2010) as the first stage of argument overlap detection. It processes a text in seven

passes; in every pass, clusters are formed by aggregating the resulting clusters of the previous pass. Attributes such as gender and number are shared within formed clusters. The passes are ordered by the precision of their methods; for example, exact match of noun phrases takes precedence over various head matches. The features they use are mostly shallow, and the system is unsupervised (i.e. not requiring training on coreference annotation), nonetheless it was already one of the best coreference resolvers at the time when the work of this thesis started (Durrett and Klein, 2013).

Of course, my system can be used with any coreference resolvers. I have experimented with the IMS HOTCoref system (Björkelund and Farkas, 2012), which is an improved version of their earlier work that outperforms the best system of the CoNLL-2012 shared task on coreference resolution. It is considerably more sophisticated than the Stanford system, mainly because it is trained to internally predict a latent tree representation of coreference clusters. Although syntactic information is also used in the Stanford system to sort candidate antecedents, it is used more flexibly as features (whose weights can be learned) in the IMS system. However, because changing to the IMS system did not show obvious improvement in summary quality, I have continued to use the Stanford system.

What can be obtained and what cannot be obtained using a coreference resolver? I will answer this question in the following two subsections.

4.1.1 Coreference annotation

Knowing that coreference means identical referents does not provide enough insights on how this principle is applied by current researchers to diverse, naturally occurring texts. A working definition of the coreference resolution task must be learned from the annotation guidelines, such as the MUC-7 Coreference Task Definition (Hirschman, 1997) and the Coreference Guidelines for English OntoNotes (Weischedel et al., 2011), which instruct the human annotators to mark or not to mark certain pairs of phrases as co-referring.

There are three goals that the guidelines have to balance: (1) the linguistic definition of coreference, (2) efficiency of human annotation and inter-annotator agreement, and (3) usefulness of the resulting annotation for downstream tasks. In general, the MUC guideline interprets coreference more broadly, but has been criticised for overextending itself (Van Deemter and Kibble, 2000). In my opinion,

it has touched the rim of argument overlap, but is confined by a more cost-effective representation. On the other hand, the later project of OntoNotes excludes some of the most disputable areas in its less sophisticated guideline.

First of all, there is the restriction on the type of expressions that are eligible to enter coreference annotation, i.e. the so-called "markables". The MUC guideline considers only noun phrases (including single-word nouns and pronouns) as markables. The OntoNotes guideline further includes verbs which co-refer with a noun phrase, for example to the same event. In both guidelines, a prenominal modifier is markable only if itself is a noun that co-refers with another noun phrase. Note that most markables are not actually marked in the resulting annotation, because both guidelines exclude singletons by requiring every coreference chain to contain two or more mentions.

The two guidelines differ in the way they treat generics. OntoNotes treats generics minimally, only including them if they are required as antecedents of a referring pronoun or definite noun phrase. For example, two mentions of the bare plural noun phrase "*cataract surgeries*" are not linked, but a mention of "*cataract surgeries*" can be linked to a pronoun "*they*" or a definite noun phrase "*those surgeries*" if they refer to "*cataract surgeries*". On the other hand, the MUC guideline contains a finer treatment of generics by interpreting them as referring to either sets or types. There is a coreference link between two generics that refer to the same set or the same type. Therefore, two mentions of "*cataract surgeries*" would be linked in MUC, in addition to the links in OntoNotes. This approach is closer to argument overlap, but of course makes annotation more difficult, because there is no surface clue to prompt the annotator that the same generic has appeared before in text. More importantly, the distinction between reference to a set and reference to a type is often not obvious, and determining whether two types are identical is a grey area that is better represented by a continuous measure.

Because coreference by definition requires the same referent, in principle there should be only one type of links, which is the identity link. The MUC guideline follows this exactly, but therefore faces a problem with what it calls "intensional descriptions". An intensional description is a predicate that is true of an entity or a set of entities. In MUC, an intensional description is linked to its extensional

description (such as a personal name or the value of a function).[4] For example, in the sentence "*Bill Clinton is the President of the United States*", there is a coreference link from the intensional description "*the President of the United States*" to the extensional description "*Bill Clinton*". Although the link between an intensional description and an extensional description may change over time, the MUC guideline requires a coreference link whenever it is expressed that the two are equivalent at any time. Consider the following example, which contains three coreference chains (numbered 1, 2, 3):

> [*Henry Higgins, who was formerly* [*sales director for* [*Sudsy Soaps*]$_2$,]$_1$]$_1$ *became* [*president of Dreamy Detergents*]$_1$. [*Sudsy Soaps*]$_2$ *named* [*Eliza Dolittle*]$_3$ *as* [*sales director*]$_3$ *effective last week.*

Chain 1 represents Henry Higgins, for whom the phrases "*sales director for Sudsy Soaps*" and "*president of Dreamy Detergents*" are included as intensional descriptions, even though the two intensional descriptions are not equivalent and are not true predicates of Henry Higgins at the same time. Furthermore, the same predicate "*sales director*" is also included in Chain 3 as a description for Eliza Dolittle. This unnatural result reflects the preference of basing annotation judgements on extensions; links between intensions are only considered as a special case in which the referent is a type instead of an entity or a set of entities.

In contrast, OntoNotes regards mentions of generics and abstract concepts as not referring at all. According to this principle, predicates such as "*sales director for Sudsy Soaps*" are attributes, and therefore are not in an equivalence relation with the referent they modify. In addition to the identity link, OntoNotes has a second type of link called the appositive link, which connects the referent to its attribute in a nominal apposition. This type of link does not apply to copular constructions, because the information of attribute predication is captured by other annotations of OntoNotes; the appositive link is in fact a supplement to that information and is not coreference per se. To determine which of the two noun phrases of an apposition is the referent (the other is attribute), precedence

[4]Of course, one intensional description can also link to another intensional description if its extensional description does not exist in the text. Although in principle the anaphoric status of a phrase is irrelevant to coreference annotation, the format of the annotation is a set of directed binary links from a mention to an antecedent-like mention. The choice of binary links within an equivalence class implies a hierarchy of representativeness which ranks extensional descriptions as more fundamental than intensional descriptions.

of types of noun phrases is defined in the OntoNotes guideline. For instance, an indefinite noun phrase is an attribute of a pronoun, regardless of the textual order of the two. This is different from its treatment of copular constructions, in which the subject is always the referent (complements are assumed attributive).

Sometimes, coreference is naturally interpreted more broadly when there is utilitarian value in doing so. One utilitarian value is substituting a less informative expression by a more informative one. For example, in the sentence "*Whenever a solution emerged, we embraced it*", both guidelines would agree to link "*it*" with "*a solution*", even though there is no referent involved according to Van Deemter and Kibble.

In summary, the coreference annotation as we know is really a projection of a presumably richer knowledge representation to the equivalence classes at one particular moment. This projection is useful for some applications, but is insufficient for argument overlap. In particular, overlaps which are based on thoughts (intensions) rather than references (extensions) are not addressed or at best addressed as secondary. The use of only one type of links is unable to represent similarity of concepts, bridging inferences (reference to an entity that is inferentially related to an antecedent, such as "*the door*" in relation to a previously introduced car), partial overlap of sets, as well as meta-statements of relations (such as possibility of identity). In my system, these issues are addressed by a continuous model of word meaning.

4.1.2 Difficulties in coreference resolution

Among all types of argument overlap that are necessary to establish local coherence, there are not only types that are excluded by the annotation guidelines, but also types that are difficult to detect by existing coreference resolution systems. Recall error is a main challenge as most systems have better precision than recall (Martschat and Strube, 2014). This justifies the two-stage set-up of my argument overlap detection, in which coreference resolution as the first stage is precision-oriented, whereas the second stage is recall-oriented.

Martschat and Strube observe that typical coreference resolvers are good at resolving anaphoric pronouns and linking co-referring names (proper nouns), but perform badly on links between a name and a common noun as well as links between two common nouns. This is not surprising given the fact that shallow

features such as distance in text, syntactic categories, gender of names (by looking up in a lexicon), and string matching are already indicative of coreference among names and pronouns. In contrast, linking a name and a common noun often requires world knowledge, and is further complicated by the language style of some of the annotated corpora such as *The Wall Street Journal* of OntoNotes. For example, in news reports, "*South Africa*" could be referred to metonymically as "*Pretoria*", and "*Apple Inc.*" could be referred to as "*the tech giant*" to increase the diversity of expressions. Linking two common nouns is also challenging, because in many cases it involves noun phrases, the meaning of which should be modelled using compositional semantics. A referring expression can be regarded as a query over entities in the reader's mind, and the most salient entity that satisfies the query is returned as the referent. In this regard, noun phrases are more complex queries than pronouns and names, because it is non-trivial to decide whether a noun phrase is semantically compatible with an entity or concept. In many implementations, the detection of semantic compatibility is done using shallow features such as requiring a subsequent mention to have the same head as its antecedent and a subset of the antecedent's modifiers.

Another reason that explains the imbalance of performance on different types of coreference is the skewed distribution of these types in corpora. As I have described in the previous subsection, the generalization of the coreference definition for generics and abstract concepts is either problematic or not attempted at all. The main obstacle is that a connection based on meaning is often a fuzzy one, instead of a clearly binary identity relation. Because the research of coreference resolution ignores any non-identical relation, many noun phrases become singletons, which are not marked in annotation. As a result, coreference resolvers are generally biased towards not linking noun phrases, and during preprocessing, likely singletons are filtered out in the same way as non-referring expressions. The sparsity of semantically-based connections means insufficient data for both training a data-driven system and demonstrating the usefulness of modelling semantics. For example, Durrett and Klein (2013) show that even a combination of several shallow semantic features fails to significantly improve a coreference resolver over a baseline of surface-level features. As an additional note, the Stanford coreference resolver does not handle event coreference and performs poorly on demonstratives, because they appear very rarely in corpora.

4.2 Distributional semantics

One common way to detect similar or related words is via distributional semantics, which is based on the hypothesis that words which are similar in meaning occur in similar contexts. Of course, this differs from how language users normally define a word, such as by relating it to other concepts, or by enumerating its members, components, or attributes. In contrast, to qualify the nature of the similarities discovered by distributional semantics is not straightforward (Lapata, 2003). However, practically speaking, distributional semantics enables us to model word meaning using mostly unsupervised methods, relying only on a corpus, whereas modelling word meaning more explicitly requires specialized data from linguists or trained annotators. Therefore, thanks for its general applicability, distributional semantics is often the preferred method for modelling word meaning in spite of its limitations. In this section, I will describe two types of distributional methods I have experimented with for modelling argument overlap.

The earliest distributional method that models the semantic distance between pairs of words is based on the notion of mutual information (Church and Hanks, 1990). In information theory, the mutual information of two random variables is a symmetric measure of the amount of information obtained about one random variable given the other random variable. We treat the pair of words in question x, y as the outcomes of the said random variables, and their pointwise mutual information (PMI) is defined as:

$$\text{pmi}(x, y) \equiv \log \frac{p(x, y)}{p(x)p(y)} \tag{4.1}$$

The frequencies of observing individual words x and y as well as both words in documents are used as estimations for $p(x)$, $p(y)$ and $p(x, y)$. Thus, pairs of words that often occur in the same document have high PMI. This simple method has been shown to perform well on some word similarity tasks (Turney, 2001).

The more advanced models of distributional semantics often represent the meaning of a word as a vector, the similarity of which can be measured in a vector space. One simple way to create a vector for word w_i is to use each dimension j to represent the frequency that a particular word w_j is observed in the neighbourhood of w_i.[5] The resulting vector is a distributional representation

[5]The neighbourhood can be defined as a context window in surface text, or more soph-

called the co-occurrence vector, and it has as many dimensions as the vocabulary size. However, it has not exploited the fact that because there is similarity between words, the dimensions corresponding to the similar words are redundant. Hence, using a low-dimensional (often 200–500 dimensions) dense vector of real numbers (called a distributed representation) can potentially enforce a better use of the data by encouraging generalization, and is less prone to suffer from data sparsity. Both methods that I use produce distributed representations.

In order to discuss what type of lexical similarity is captured by these models, let me introduce two notions of word association that are commonly used in psychology and linguistics: paradigmatic association and syntagmatic association. Paradigmatically similar words stand for concepts of the same type, for example "*apple*" and "*orange*". They are thus called because they can often fit into the same syntactic or semantic role. In contrast, syntagmatically similar words are related to the same topic, and are not necessarily of the same type, for example "*wind*" and "*weather*". Syntagmatic similarity is sometimes called relatedness, when the notion of "similarity" is narrowed to paradigmatic similarity. It is generally agreed that the size of the context window influences the type of similarity that is captured by the resulting vector representations: larger context windows correspond to more emphasis on syntagmatic similarity (Levy and Goldberg, 2014). Methods that are based on word-document relations instead of word co-occurrences (such as LSA, which I will discuss later) can be regarded as having a very large context window, and hence mainly capture syntagmatic similarity.

As I have demonstrated by examples at the beginning of this chapter, argument overlap is related to both types of similarity. In fact, which type is preferred by a human reader is influenced by many factors. The potential problem of using either type of similarity for argument overlap is that only a small portion of the use of similar terms can be regarded as near-synonyms or paraphrases that refer to the same concept. In the case of paradigmatic similarity, many words under the same type represent mutually exclusive concepts, which should be treated as unrelated or even contrasting when they occur in the same text. On the other hand, syntagmatically related words are naturally abundant in a coherence discourse; if they are indiscriminately regarded as representing the same concept,

isticatedly in terms of typed grammatical dependencies (Lin, 1998). In the latter case, each dimension captures the frequency of the word being under a particular grammatical relation with a particular word.

propositions about these words would be deprived of their information content.

Intuitively, high strengths of overlap should only be assigned to near-synonyms, i.e. expressions that represent confusable concepts in that context. An important condition for near-synonymy is the lack of conflicts in the meanings of the two words. This intuitive notion of semantic compatibility, however, cannot be perfectly measured by a simple combination of the two types of similarity. As for other similar words, a lower strength of overlap should be chosen in such a way that attachment via intermediate propositions is preferred over attachment via these non-identical relations. In other words, because two words are regarded as mostly separate concepts, it is preferred to use any relation of them given by the text, rather than an underspecified "similarity" relation, which can be regarded as an implicit proposition that connects the two words. These weak overlaps are secondary connections, used for attachment only in case when other possibilities are also weak.

I have experimented with two representative methods of distributional semantics to study how well they approximate the desired properties of the overlap function. The first one is a widely used topic modelling technique called latent semantic analysis. The second one is the more recent development of learning word representations using neural networks. (By convention, the distributed representations learned this way are called word embeddings, although by definition word embedding can refer to any distributed representation.) I will now explain these two methods in the following subsections.

4.2.1 Latent semantic analysis

Landauer et al. (1998) developed latent semantic analysis (LSA) as a method to automatically induce human conceptual knowledge from text data. Instead of acquiring knowledge directly from perceptual information, instinct, or experiential intercourse, they were interested in the potential of inducing knowledge via documents, which are regarded as episodes of life and arguments. The intuition of LSA is that, in contrast to using only first-order word co-occurrences, it models the contexts in which a word does or does not appear as a latent variable (which can be construed as topic).

The development of LSA was closely related to the detection of coherence and simulations of human comprehension, which makes it an ideal candidate for

solving argument overlap. For example, Foltz et al. (1993, 1998) applied LSA to calculate textual coherence, and show that coherence evaluated this way correlates well with several performance indicators of human comprehension. Interestingly, Bestgen et al. (2010) found a negative correlation between LSA-based coherence and human grading of essays by foreign language learners, but which may be explained by the preferences of the grading criteria (for instance favouring high lexical diversity). Kintsch (1988) also uses LSA in his CI model, which I have introduced in Chapter 2.

The first step of LSA is to construct a term–document matrix M, in which each row corresponds to a term and each column to a document. Each cell M_{td} represents the frequency of term t in document d, usually normalized by some tf–idf method. In addition to the conventional tf–idf method I have introduced in Chapter 2, there is a different weighting method in Landauer et al.'s paper: The value of a cell is the logarithm of the term frequency (+1) in that document, divided by the entropy of the distribution of documents given that term:

$$M_{td} = \frac{\log(freq_{td} + 1)}{H(D|t)} \tag{4.2}$$

When calculating the entropy of a term, the distribution of documents is estimated using the term's frequency in every document:

$$H(D|t) = -\sum_{d} p_{td} \log p_{td}, \text{ where } p_{td} = \frac{freq_{td}}{\sum_{d} freq_{td}} \tag{4.3}$$

Terms that have almost uniform distributions of documents will have high entropy, and thus are discounted in M. Because this method has been shown to be superior to several alternatives including the conventional tf–idf (Nakov et al., 2001), I use this method in my experiments.

The second step is to compute the singular value decomposition (SVD) of matrix M. The SVD of a real matrix M is the product of three matrices USV^T, in which U and V contain the eigenvectors of MM^T and M^TM respectively, and the diagonal entries of S are the square roots of the eigenvalues of both MM^T and M^TM in descending order. By keeping only the first k eigenvalues in S and truncating the corresponding columns of U and V, the product USV^T reconstructs a least-squares approximation of the original M. If the original M is of size $m \times n$

(m terms, n documents), the dimensionality-reduced U and V would be of size $m \times k$, and the reduced S would be $k \times k$, where usually $k \ll n$. In this compact representation, the rows of U represent the terms as k-dimensional vectors. One measure of similarity between two terms i and j is the cosine similarity of $w_i = Su_i$ and $w_j = Su_j$, in which u_i and u_j are rows i and j of U, transposed into column vectors:

$$\text{similarity}_{\cos}(w_i, w_j) \equiv \frac{w_i \cdot w_j}{\|w_i\|\|w_j\|} \tag{4.4}$$

Preprocessing a corpus prior to LSA can provide better modelling of words from noisy data. The preprocessing stage in my experiment includes normalizing text to lowercase and removing diacritics. In order to model concepts that are expressed as phrases, I also merge multi-word expressions into single tokens. The way this is achieved is by using a data-driven, unsupervised algorithm by Mikolov et al. (2013b), which scores the collocation of two tokens i and j as follows:

$$\text{score}(i, j) = \frac{\text{frequency}(i, j) - \delta}{\text{frequency}(i) \times \text{frequency}(j)} \tag{4.5}$$

In this equation, frequency(i) and frequency(i, j) are the frequencies of the unigram i and the bigram i, j in the corpus, and δ is the minimum frequency of a valid unigram or bigram. Bigrams that have higher scores are more likely to be idiomatic expressions. After merging the highest scoring bigrams (e.g. "*da_vinci*"), the algorithm can be repeated on the processed corpus to identify longer expressions (e.g. "*leonardo_da_vinci*"). The algorithm could fail to identify an expression when it includes a token which is highly frequent by itself. For robustness, I use the implementation in the `gensim` package (Řehůřek and Sojka, 2010), which allows ignoring a user-defined set of words. By ignoring the stopwords defined by `nltk` (Bird et al., 2009), expressions such as "*house_of_representatives*" (containing the stopword "*of*") can now be identified.

4.2.2 Word embeddings

In addition to LSA, in which the context information of a word is first collected and then compressed, vector representation of word meaning can also be learned in a single step as a by-product of predictive tasks. In the literature, these two types of methods are known as counting models and predicting models (Baroni et al.,

2014). The premise of predicting models is transfer learning. That is, a word representation optimized for a simple predictive task should have encoded some aspects of the meaning of that word, which make it a useful representation for other tasks as well. In particular, words that have similar properties are mapped to continuous vectors (called embeddings) that are close to each other.

There are many tasks from which word embeddings can be derived, including tasks that correspond to real applications as well as artificial tasks that are designed only as a means to learning embeddings. Because different tasks may depend on different word properties, obviously it would be ideal to obtain embeddings specifically optimized for the task where they are used. However, many tasks, including summarization or the simulation of comprehension, have too little training data and are computationally costly as compared to artificial tasks. Therefore, it is common in practice to use embeddings derived from an artificial task, either directly or as the starting point for further training.

One important type of artificial task is called word2vec (Mikolov et al., 2013a), which has two variants in the task definition: the continuous bag-of-words (CBOW) model, and the skip-gram model. In the CBOW model, the task is to predict the occurrence of a word given its context (words immediately before or after it), whereas in the skip-gram model, the prediction is in the opposite direction. The architecture of word2vec models is a simple feedforward neural network consisting of an input layer, a projection layer, and an output layer. In both the input and the output layers, a word occurrence is represented as a vector in which each element is the probability of a particular word in vocabulary. Traditionally, the probability distribution of an output word is estimated using the softmax function, which requires iterating through the entire vocabulary; but techniques such as hierarchical softmax and negative sampling have enabled efficient learning by only examining a small fraction of the vocabulary at once.

An other task for learning word embeddings is the GloVe model (Pennington et al., 2014), which lies on the border between counting models and predicting models. Like LSA, it explicitly collects co-occurrence statistics, but the type of statistics is term–term co-occurrences instead of term–document relations. The co-occurrence statistics is not used for matrix factorization, but for defining an optimization objective. In particular, the loss function is defined as the (weighted) squared difference between the dot product of the vectors of two words, and the logarithm of their (weighted) co-occurrence count in the training corpus. Like

neural network models, the word vectors that minimize the loss are found using gradient-based optimization.

As I have explained at the beginning of this section, the detection of argument overlap awards a special position to paraphrasing or near-synonymy relations. However, using distributional methods, it is often difficult to distinguish near-synonymy from other similarities, because words of the same type or topic can also occur in similar contexts, even if they have incompatible attributes or represent disjoint sets of things. To learn word embeddings that are sensitive to semantic compatibility, we could set the optimization goal to minimizing the distance between terms that can often be used to stand for the same concept. There are different ways to harvest paraphrases,[6] and one resource-rich area is translation. As there are often multiple possible translations for one term in the source language, these alternative formulations are paraphrases of each other in their context.

In neural machine translation, the objective of letting alternative formulations have similar embeddings is enforced automatically in the training process. For example, in the model proposed by Cho et al. (2014), a source sentence is represented as an order sequence of word embeddings of the source language; a recurrent neural network (RNN) encoder encodes this sequence of embeddings into a fixed-length vector, which is then decoded by an RNN decoder word by word into the target language. If multiple terms in the source language frequently correspond to the same term in the target language, the embeddings of these terms have to be close to each other to ensure they have similar impacts on the encoded representation. This effect is made more explicit by Bahdanau et al. (2015), whose model additionally learns to predict a soft alignment between the source sentence and the target sentence. Hill et al. (2015) evaluated the word embeddings derived from two Bahdanau et al. models trained for English–French and English–German translation, and found the resulting embeddings for English to perform well on many semantic tasks. In my experiments, I will test whether these embeddings lead to better modelling of argument overlap by evaluating the summarization performance.

[6]For example, on Wikipedia, various expressions of the same concept can be extracted from the descriptions of the links to the same article.

4.3 Lexical chains

Instead of modelling the meaning of individual words and phrases, a different perspective for the question of argument overlap is to model the topic progression of a discourse. A lexical chain (Morris and Hirst, 1991) is a sequence of related expressions found in the text whose presumed senses in context are related to the same concept or topic. If two propositions contain arguments that are in the same lexical chain, then the two propositions are assumed to be a coherence relation, such as the later proposition being an elaboration of the previous one. Therefore, the attachment of propositions can be based on a combination of coreference chains and lexical chains.

Barzilay and Elhadad (1999) apply lexical chains to summarization, considering them to be more robust than using cue phrases and text location in detecting coherence relations. To produce the summary of a text, they first rank its lexical chains in descending order of importance, and then realize them one by one, extracting one representative sentence per chain. The importance of a chain is determined by its length and the number of distinct word types. The representative sentence of a chain is the first sentence in text order which contains, by different heuristics, either any of its members, or a member whose word type is no less frequent than the average in that chain.

In contrast, I do not rank propositions directly based on lexical chains, but use them as a source of information for the detection of argument overlap. The role of lexical chains is to supplement coreference chains, which are mostly equivalence classes of specific entities, with information about the topicality of the general concepts involved. There are two advantages of lexical chains over distributional models in fulfilling this role:

1. Because word senses are disambiguated by context in the process of computing lexical chains, spurious overlaps caused by interpretations that cannot possibly be true at the same time can be prevented. For example, consider Figure 4.2, which shows three subtrees in the working memory, containing propositions that correspond to the text pieces (1) "*[fire was] a gift randomly delivered in the form of lightning, forest fire or burning lava*", (2) "*fire-lighting was revolutionised by the discovery of the element*"[7], and (3)

[7]Let us assume that the coreference resolver fails to find a coreference chain for "*the element*", and therefore in the proposition it is not substituted by its antecedent.

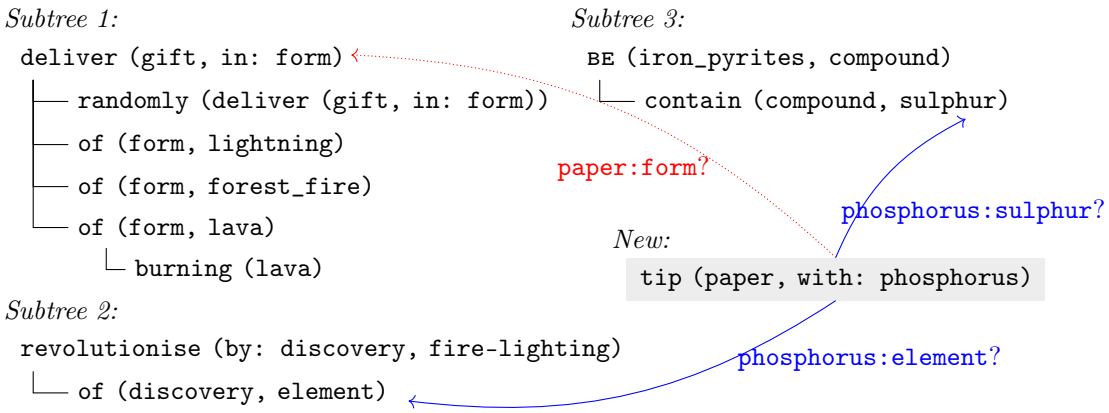


Figure 4.2: Possible attachments of a new proposition.

"*iron pyrites, a compound that contains sulphur*", respectively. The new proposition corresponds to the text "*paper tipped with phosphorus*". It can attach in subtree 2, because "*phosphorus*" is a kind of "*element*"; it can also attach in subtree 3, because both "*phosphorus*" and "*sulphur*" are chemical elements, although this connection is comparatively less plausible. However, because "*paper*" and "*form*" are found similar by distributional semantics (owing to a sense of "*form*": "`form/8` – a printed document with spaces in which to write"), the simulation may be misled into attaching the new proposition as a child node of the root proposition of subtree 1 due to the higher tree level. This mistake could happen even in the presence of other attachments that are based on a different sense of "*form*", because globally a word is not constrained to one sense.

2. Because lexical chains are constructed using a thesaurus, which marks word relations such as synonymy and hypernymy, it is possible to offer different treatments to different word relations. In particular, synonymy is associated with a very strong overlap (i.e. identity relation); hypernymy is weaker as it may indicate a shift within the same topic. The argument overlap between siblings which share a hypernym is even weaker, as such siblings are often (but not always) disjoint concepts (e.g. "*sulphur*" and "*phosphorus*" as mentioned above). In a distributional model, however, different types of word relations are mixed together, and it is hard to come up with a threshold that effectively separates compatible and incompatible relations. The thesaurus that is commonly used is WordNet (Miller, 1995), which

represents every concept as a synset (synonym set). A potential drawback of using a thesaurus is insufficient coverage of words, but in my experiments WordNet covers 98.3% of all word occurrences to be processed by lexical chains, excluding those identified as proper nouns.

My implementation of lexical chains is mainly based on Galley and McKeown (2003). Their work improves over previous methods by enforcing the rule that all occurrences of the same word should take the same sense in a document. This is a very strong assumption and may not always hold, nevertheless it improves word sense disambiguation accuracy, by taking more context (the entire document, instead of case-by-case local contexts and decisions) into consideration when determining the sense of one word and hence reducing the influence of noise. Their method has been shown to improve summarization quality as well (Ercan and Cicekli, 2008).

4.3.1 Building lexical chains

I create lexical chains in two steps. First, a disambiguation graph is created, in which the vertices are word occurrences (represented as identifiers consisting of sentence and token numbers) collected from all propositions of the document. An edge in the graph represents a possible relation between two arguments, and has the following attributes: the lexical relation between the two arguments, the presumed word sense of each argument on which the lexical relation is based, as well as a weight. Because a pair of arguments may be related under different presumptions of their word senses, there can be multiple edges between two vertices, hence the graph is a multigraph. Second, I select one word sense for each word, and remove from the disambiguation graph any edge that is based on an unselected sense of either of its two ends. In this disambiguated graph, there is now at most one edge between two vertices. The sum of weights of the graph's remaining edges is the score of this interpretation, and the connected components of the graph correspond to the lexical chains. The goal of the sense-selection algorithm is to find the highest-scoring interpretation.

I keep an auxiliary data structure for looking up lemmas of word occurrences, looking up senses (synsets) of lemmas, and reverse lookups. This data structure is a tripartite graph, in which edges exist only between a word occurrence and a lemma, or between a lemma and a synset. The lemma of a word occurrence

is what I previously referred to as "word"; occurrences of the same lemma are regarded as instances of the same type. The lemma of a multi-word expression is the longest sub-sequence of the expression that contains its head word and is an entry of WordNet. Following the example of Silber and McCoy (2002), I include a pseudo-sense for every proper noun (identified by part-of-speech tagging), in addition to any sense found in WordNet, so that out-of-vocabulary named entities may form a lexical chain based on a pseudo-sense. Because the lemma of a pseudo-sense is just the string of the word or phrase occurrence, which may not be identical to the lemma of real synsets, such an occurrence can map to at most two lemmas.

4.3.1.1 Step 1: Adding edges

In the first step, I iterate through every word occurrence (argument or functor) of every proposition, and add it to the graph if it has not already been processed and is a noun or verb. When adding a new word occurrence, for every synset of its lemma, the following synsets are inspected: the synset itself, its hypernyms (including instance hypernyms), its siblings (synsets which share a hypernym wit it), its hyponyms (including instance hyponyms), and derivationally-related verbs of a noun. Using the auxiliary data structure, I can look up all processed word occurrences which are mapped to these synsets, and create an edge between the existing vertex and the new vertex in the disambiguation graph. The entire process for common nouns is detailed as Algorithm 6; the process for verbs and proper nouns is similar and can be handled by additional conditions to this outline. The relatively complex way to retrieve derived word forms (Lines 23 to 32) is due to the fact that a WordNet synset can have multiple "lemma" objects (and only one of them is the lemma in question), with which the derived forms are associated.

The weights of edges follow Galley and McKeown (2003), but with the addition of verbs. Verbs are added for two reasons: to model nominalized events and to provide a higher number of connections. Having too few connections would otherwise be problematic, as unlike in the distributional semantics models, words have to be in the same lexical chain to have non-zero overlap values. Having more connections also benefits the disambiguation process, whose accuracy depends the number of data points.

Algorithm 5 Helper procedure for Algorithm 6. Edges in G which involve the same pair of vertices are distinguished by their keys, which are pairs of synsets (x, y) representing the senses the vertices have to take if the edge is finally selected after disambiguation. The word occurrence (vertex) which occurs first in text corresponds to synset x, whereas the other vertex corresponds to synset y.

Require: G, the disambiguation graph
Require: g, the auxiliary tripartite graph
Require: w, a word occurrence
Require: s_w, the presumed sense of w
Require: s_v, a sense to which s_w is related
Require: *relation*, the lexical relation under which s_w and s_v are related
Ensure: an edge is created in G between w and any other node in G that has a possible sense of s_v

1: **procedure** AddEdges($G, g, w, s_w, s_v, relation$)
2: **for all** $v \in$ word occurrences in g which have a two-edge path to s_v except w **do**
3: **if** $v < w$ **then** ▷ word v occurs before w in text
4: create edge in G between w and v with key (s_v, s_w) and call it e
5: **else** ▷ word v occurs after w in text
6: create edge in G between w and v with key (s_w, s_v) and call it e
7: **end if**
8: assign weight to e by looking up Table 4.2 according to *relation* and the textual distance between w and v
9: **end for**
10: **end procedure**

Algorithm 6 Algorithm to create the disambiguation graph, assuming that w is a noun. The subroutine AddEdges is defined in Algorithm 5. Special flags (constants) are written in small caps, e.g. SYNONYMY. A statement of "create edge" also adds the involved vertices to the graph if not already present.

Require: *propositions*, a list of propositions
Ensure: the disambiguation graph is created, containing word occurrences in *propositions* with weighted edges

```
 1: function CreateDisambiguationGraph(propositions)
 2:     G ← empty multigraph                          ▷ the disambiguation graph
 3:     g ← empty graph                               ▷ the auxiliary tripartite graph
 4:     for all p ∈ propositions do
 5:         for all w ∈ functor and arguments of p do
 6:             if w is a node in g then
 7:                 continue                          ▷ w is already processed
 8:             end if
 9:             l ← lemma of w as text string                              ▷ cf. page 91
10:             create edge in g between w and l
11:             for all s_0 ∈ synsets of l do
12:                 create edge in g between l and s_0
13:                 AddEdges(G, g, w, s_0, s_0, SYNONYMY)
14:                 for all s_1 ∈ hypernym synsets of s_0 do
15:                     AddEdges(G, g, w, s_0, s_1, HYPERNYMY)
16:                     for all s_2 ∈ hyponym synsets of s_1 except s_0 do
17:                         AddEdges(G, g, w, s_0, s_2, SIBLING)
18:                     end for
19:                 end for
20:                 for all s_3 ∈ hyponym synsets of s_0 do
21:                     AddEdges(G, g, w, s_0, s_3, HYPERNYMY)
22:                 end for
23:                 for all l_1 ∈ WordNet lemma of s_0 do
24:                     if the name of l_1 is l then
25:                         for all l_2 ∈ derived forms of l_1 do
26:                             s_4 ← the synset of l_2
27:                             if s_4 is a verb then
28:                                 AddEdges(G, g, w, s_0, s_4, DERIVATION)
29:                             end if
30:                         end for
31:                     end if
32:                 end for
33:             end for
34:         end for
35:     end for
36:     return G
37: end function
```

	Lexical relation			Textual distance			
Type	Noun	Verb	Derivation	1 sent.	3 sent.	1 par.	other
0	synonymy	synonymy	noun–verb	1	1	0.5	0.5
1	hypernymy	hypernymy	–	1	0.5	0.3	0.3
2	sibling	sibling	–	1	0.3	0.2	0

Table 4.1: Configuration LC-0: weights of edges

	Lexical relation			Textual distance			
Type	Noun	Verb	Derivation	1 sent.	3 sent.	1 par.	other
0	synonymy	–	–	1	1	0.5	0.5
1	hypernymy	synonymy	noun–verb	1	0.5	0.3	0.3
2	sibling	hypernymy	–	1	0.3	0.2	0

Table 4.2: Configuration LC-1: weights of edges

Like Galley and McKeown, the weight of an edge is based on both the lexical relation and the textual distance (in number of sentences or paragraphs) between the two word occurrences. I have created four configurations, LC-0 (Table 4.1), LC-1 (Table 4.2), LC-2 (Table 4.3), and LC-3 (Table 4.4), which differ in the status of verbs and derived forms. For instance, in LC-1, the weight of verbs is downgraded with respect to nouns; for a pair of verbs to achieve a certain weight, they need to be closer in text or in a stronger lexical relation than a pair of nouns with the same weight. I will later show in an experiment (Subsection 5.3.2) that LC-1 gives better performance than LC-0, LC-2, and LC-3.

To illustrate how the algorithm works, let us consider a small example involving four propositions extracted from a real text (Table 4.5). For simplicity, we only consider the underlined word occurrences. Processing the first proposition, the three possible senses of the noun `compound` are added to the auxiliary graph (Line 12); nothing is added to the disambiguation graph because this is the first word we process. In the next proposition, the noun `chemical` has only one sense "`chemical/1` – material produced by or used in a reaction involving changes in atoms or molecules", which is the hypernym of one of the sense of `compound`: "`compound/2` – (chemistry) a substance formed by chemical union of two or more elements or ingredients in definite proportion by weight". Therefore, an edge is created between `compound` and `chemical` in the disambiguation graph; its weight is 0.3 because they are in a hypernymy relation and they are more than one paragraph apart. In the third proposition, the noun `product` has a sense "`product/4` – a chemical substance formed as a result of a chemical reaction", which is a hyponym

Lexical relation				Textual distance			
Type	Noun	Verb	Derivation	1 sent.	3 sent.	1 par.	other
0	synonymy	–	–	1	1	0.5	0.5
1	hypernymy	–	–	1	0.5	0.3	0.3
2	sibling	synonymy	noun–verb	1	0.3	0.2	0

Table 4.3: Configuration LC-2: weights of edges

Lexical relation				Textual distance			
Type	Noun	Verb	Derivation	1 sent.	3 sent.	1 par.	other
0	synonymy	–	–	1	1	0.5	0.5
1	hypernymy	–	–	1	0.5	0.3	0.3
2	sibling	–	–	1	0.3	0.2	0

Table 4.4: Configuration LC-3: weights of edges

of `chemical/1` and hence a sibling of `compound/2`. Similarly, an edge of weight 0.3 is added to the disambiguation graph between `product` and `chemical`, but no edge is added between `product` and `compound` because the weight is 0 here. Finally, the last proposition adds three edges, each of weight 0.5, between `product` and `produce`, because the verb `produce` is a derivationally-related form of three different senses of the noun `product`.

The resulting disambiguation graph is shown in Figure 4.3. It is the task of the second step to commit each word to a particular sense, and keep only the edges which are consistent with that interpretation. In this example, we would want `product` to take sense `1`, `2`, or `5`, rather than `4`, so that two lexical chains will result (`{compound, chemical}` and `{product, produce}`). I will now show how this is achieved by optimization.

Paragraph	Sentence	Proposition
4	3	`contain (compound, sulphur)`
6	4	`treat (splint, with: chemical)`
8	1	`market (Samuel_Jones, product)`
	2	`produce (student, match)`

Table 4.5: Example propositions with the number of paragraph/sentence where they are taken from

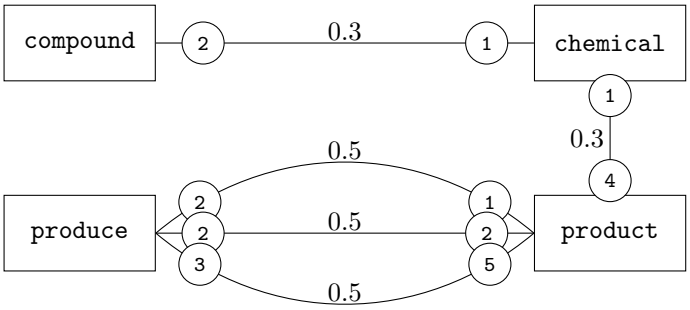


Figure 4.3: Disambiguation graph for the underlined words in Table 4.5. The circled numbers on the edges indicate the senses the words on both ends should take if the edge is valid.

4.3.1.2 Step 2: Disambiguation

In the second step, there are two possible ways to select the sense for each lemma: by using integer programming, or by a greedy algorithm. Galley and McKeown used the greedy algorithm, which of course does not guarantee to maximize the total weight of the remaining edges in the disambiguation graph after disambiguation. Here, I present for the first time a formulation that uses quadratic programming to maximize the total weight: Let each variable x_i be a boolean value representing whether a particular lemma is assigned with a particular sense. The constraint that for each lemma there is exactly one variable that has value 1 can be expressed as a linear constraint $Ax = b$, in which x is the vector of variables, each row of A corresponds to a lemma, and b is a vector of 1s. The mathematical formulation is the following:

$$\text{maximize} \qquad x^T Q x \tag{4.6}$$

$$\text{subject to} \qquad Ax = b \tag{4.7}$$

In the objective function, Q is a symmetric matrix of weights; Q_{ij} is the total weight of all edges in the disambiguation graph which depend on both assignments x_i and x_j being true. I use the IBM CPLEX optimizer to solve this program. In the case of the example above (Figure 4.3), the constraint $Ax = b$ is the following:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_{\texttt{compound/2}} \\ x_{\texttt{chemical/1}} \\ x_{\texttt{product/1}} \\ x_{\texttt{product/2}} \\ x_{\texttt{product/4}} \\ x_{\texttt{product/5}} \\ x_{\texttt{produce/2}} \\ x_{\texttt{produce/3}} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.8}$$

and the weight matrix is the following:[8]

$$Q = \begin{bmatrix} 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \end{bmatrix} \tag{4.9}$$

Alternatively, the sense of each lemma can be determined greedily, by selecting the sense that maximizes the sum of weights of the edges connected to all occurrences of its lemma. It is easy to show this method is not optimal in terms of the above-mentioned objective, but in practice the resulting lexical chains are mostly identical to those in the optimal solution. Therefore, considering both model plausibility and computational efficiency, I use the greedy method in the experiments in Chapter 5. In the example above, the senses `compound/2`, `chemical/1`, `product/1`, and `produce/2` will be selected (there is a tie for `product`, and the most common sense, i.e. the sense with the smallest number, is selected). This solution turns out to also achieve the maximum score, which is $0.3 + 0.5 = 0.8$. Here, the success is due to the fact that the erroneous edge between `chemical` and `product` is weaker than the correct edges between `produce` and `product`. In a real situation, there are far more words to consider, thus coincidental connections

[8]Note that while in this example Q happens to be the adjacency matrix of the disambiguation graph, this is due to the fact that each lemma has only one word occurrence. In the general case, the values in Q are derived by summation, as stated above.

are overpowered more easily.

4.3.2 Using lexical chains

Given the lexical chains, how should the strength of overlap between two arguments be calculated? Instead of assigning an overlap value of 1 uniformly to any pair of arguments in the same chain, I quantify the overlap by the semantic transitions between the two ends, which correspond to the shortest path in the disambiguation graph. For this purpose, the textual distance between occurrences is irrelevant to edge distance (unlike edge weight), but which only depends on lexical relation. The distance of an edge $d_e = a^{-t}$, in which t is the type number of the lexical relation in Table 4.2, and a is an attenuation factor between 0 and 1. The strength of overlap is the reciprocal of the shortest distance $\frac{1}{\sum_{e \in P} d_e}$ (where P is the shortest path). The intuition is to regard the semantic transitions required to make the connection as hidden, implicit propositions inside a tree attachment. Hence, more semantic transitions make the attachment via this pair of arguments less favourable, in a way comparable to attaching at a lower tree position. I set the value $a = 0.7$, meaning that a simple transition such as via a noun hypernymy relation is counted as less than a tree level, but more complex transitions are penalized more heavily. I found empirically that the use of this graded overlap as well as the introduction of verbs into the lexical chains have a positive influence over the summarization results.

As a final note, there is a possibility to improve lexical chains, in particular to address the potential sparsity of context information even more than the one-sense-per-discourse constraint does. In the previous subsection, I motivated the inclusion of verbs by the benefit of more connections, because having too few edges means the disambiguation of a lemma has to be based on the few connections the lemma has, which may be coincidental. To make word sense disambiguation in lexical chains more robust, more context information has to be added to also influence the choice of sense for each lemma. For example, one may combine the current approach with the Lesk method (1986), which in the simplest form detects if any surrounding word is used in the dictionary definition of a sense. One can also use the output of a supervised word sense classifier, such as Zhong and Ng (2010), as additional information about which sense is more probable.

Furthermore, the lexical relations that amount to edges in a disambiguation

graph can be extended beyond direct hypernymy and sibling relations. This not only provides more context information, but may also help to increase the coverage of non-zero overlap values. For example, `chemical` is a third-degree hypernym of `sulphuric_acid` (via `compound` and `acid`), which renders an edge between the two words impossible. Under the current method, the two words can be in the same lexical chain if they are indirectly connected via occurrences of `compound` and `acid`, which is already a relaxation from classical approaches which require words in the same chain to be closely connected to a central synset. A better method would require replacing the lookup table of weights (Table 4.2) by a function that computes weight according to textual distance and a notion of WordNet path similarity (such as that by Jiang and Conrath (1997)).

Chapter 5

Experiments

In the previous chapters, I have described the creation of a summarization system based on manipulation of propositions. In this chapter, I will evaluate this summarizer for two purposes: to test the impact of different modules (such as different models of argument overlap), and to compare the performance of the summarizer to other existing summarizers. Ideally, such a proposition-based summarizer should be evaluated based on the summary propositions it outputs. However, evaluation methodologies based on proposition-like meaning units are not yet commonplace. Instead, I evaluate my summarizer by generating textual summaries. Therefore, in this chapter, I will also demonstrate different methods of summary text generation, which at the same time is important to the usability of the summarizer.

The way I evaluate summaries will be presented in Section 5.1. The two methods, one automatic and the other manual, are both quantitative comparison to human-written gold standard summaries. I have created a corpus of texts and summaries (Section 5.2), which will be used in all experiments of this chapter. In the first group of experiments (Section 5.3), extractive generation is used to create summary texts from summary propositions. Although extraction is not an ideal generation method for proposition-based summarization, as the resulting texts only indirectly reflect the selected propositions, it is a relatively objective (not complicated by generation techniques) and effective way to test the content selection ability. The second group of experiments (Section 5.4) concerns an abstractive generation module, which was later added to the system directly realize summary propositions into natural language sentences. I will compare

this proposition-based generation to the common paradigm of pipeline systems consisting of sentence extraction followed by sentence compression, as well as an end-to-end neural network-based abstractive summarizer.

The work presented here has been published in three venues. In Fang and Teufel (2014), I present a prototype of my system and detailed my implementation of the KvD model. The effectiveness of my innovations, such as root change in the memory cycle, and distributional similarity for modelling argument overlap, is tested, but only on a very small dataset. After the creation of my own dataset, I present updated results in Fang and Teufel (2016).

In Fang and Teufel (2016), I mainly focus on different models of argument overlap. The experiments presented in this paper are a subset of the experiments in Section 5.3. During the write-up of this thesis, I reimplemented the entire system, which has caused some experiment results to differ slightly. I will make remarks about the differences in that section. The main update in the new implementation is in the proposition building module, which previously was based on the Stanford collapsed dependencies with propagation of conjunct dependencies, but now uses basic dependencies only. That is to say, the new version reflects my thinking of which propositions and arguments should be distributed, whereas the previous version delegated this task to the algorithm of the Stanford parser.

The summarizer with abstractive generation (Fang et al., 2016) is my collaborative work with several co-authors. I will attribute contributions to my co-authors where applicable when I describe the experiments in Section 5.4. Due to the collaborative nature of the work, these experiments have not been redone using the new implementation.

5.1 Evaluation methodology

Evaluating the quality of summaries is a difficult problem. First of all, there is not one unique criterion of summary-worthiness. Human readers' assessment of the same summary may differ because they perceive the value of pieces of information differently by assuming different backgrounds and purposes, and because they prefer different language styles or ways of structuring information. To obtain a reliable human evaluation, each summary has to be assessed by multiple readers. Furthermore, human judgement cannot be obtained immediately

as configurations change or as a model is being trained. Therefore, an automatic method of evaluating summaries is necessary for researchers to test different hypotheses and experiment with alternative modules.

In this chapter, I mainly evaluate summaries by comparison to human-written gold standard summaries (also called reference summaries), although there is a small experiment based on direct human judgement reported in Section 5.4. The comparison can be done automatically or manually. In the following subsections, I will introduce the methods I use for automatic and manual evaluation.

5.1.1 ROUGE

ROUGE (Lin, 2004) is widely-used automatic tool which computes the textual overlap between a system summary and a reference summary. Although the way it measures text similarity is shallow, it has been shown to correlate reasonably with human judgements. More importantly, it includes several different measures, some of which mainly reflect content selection, while some others place more weight on the quality of language. Although it is impossible to completely decouple content selection from linguistic realization in a text-based evaluation, different ROUGE measures enable us to evaluate a summarizer from slightly different perspectives.

In the following experiments, I report four ROUGE metrics: ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-SU4.

- ROUGE-1 and ROUGE-2 are the recall of unigrams and bigrams of a reference summary. When there are more than one reference summary, the reported score is the average over all reference summaries. In order to handle length variations in summaries (longer summaries can achieve higher recall easily), the F-score is reported instead of the recall. But according to my observation, text length still has a huge influence on the F-score, and therefore a strict control of word count is enforced (I will explain how this is done later).

- ROUGE-L is based on the notion of longest common subsequence (LCS), by treating a sentence as a sequence of words. A sequence $[z_1, z_2, ..., z_n]$ is a subsequence of another sequence $[x_1, x_2, ..., x_m]$ if there exists a strictly increasing sequence of indices $[i_1, i_2, ..., i_n]$ such that for all $j = 1, 2, ..., n$, $z_j = x_{i_j}$. The union LCS of a sentence r_i with a set of sentences C (denoted as $LCS_{\cup}(r_i, C)$) is defined as the longest subsequence of r_i that can be construc-

ted from the set of individual LCSs between r_i and each $c_j \in C$. For example, suppose $r_i = [w_1, w_2, w_3, w_4, w_5]$, and C only contains two sentences $c_1 = [w_6, w_7, w_1, w_8, w_2, w_9]$ and $c_2 = [w_2, w_9, w_3, w_4]$, then $LCS(r_i, c_1) = [w_1, w_2]$ and $LCS(r_i, c_2) = [w_2, w_3, w_4]$, therefore $LCS_{\cup}(r_i, C) = [w_1, w_2, w_3, w_4]$.

To compute ROUGE-L, for every sentence in the reference summary, we find the length of its union LCS with all sentences in the system summary.[1] Then we sum up these lengths, and divide this sum by the length of either text to compute the recall or the precision, and finally the F-score is reported.

- ROUGE-SU4 uses the skip-bigram statistics, in addition to unigrams (as indicated by the letter "U" in the name). The match of a skip-bigram is the occurrence of a pair of words in their sentence order but can be non-consecutive (separated by other words). The number 4 in the name of the metric is the maximal gap for a skip-bigram to be counted. In this case, a sequence of 7 words has $\binom{7}{2} - 1 = 20$ skip-bigrams, because only the skip-bigram consisting of the first word and the last word is too far apart to be counted. The F-score is based on the recall and precision computed by dividing the number of matches by the total number of skip-bigrams in either text.

As the experiment results later in this chapter will show (Subsection 5.4.3), ROUGE-1 and L mainly reflect content selection, which can be explained by their relatively relaxed criteria of matching. In contrast, ROUGE-2 and SU4 are more sensitive to the quality of language, and tend to penalize imperfections in text generation.

ROUGE is a surface method, but arguably there is at least one factor that makes it less inappropriate in my situation. When creating the evaluation corpus, which I will describe in the next section, the human summarizers were asked to use the original wording whenever possible. For extractive summarizers, which have no paraphrasing ability, this makes the recall of n-grams be as close as possible to the actual recall of content. The possible exception is the abstractive summarizers, which are evaluated in Section 5.4. However, as I will show in that section, their use of paraphrases is still quite limited. In the future, when more advanced abstractive summarizers become available (for example, one that does

[1] In the latest version of ROUGE, which I use here, every system sentence can only contribute to the union LCS of one reference sentence.

not only rank existing propositions but also creates new ones by inference, i.e. macropropositions), evaluation methods based on content units will eventually prove more useful than token-based methods for their deep analysis. In the next subsection, I will describe one such method, which I use in addition to ROUGE in Section 5.4.

5.1.2 The pyramid method

The most sophisticated evaluation method is the pyramid method (Nenkova and Passonneau, 2004). It breaks down both system-generated summaries and reference summaries into Summarization Content Units (SCUs), which resemble van Halteren and Teufel's (2003) factoids. The first step is the creation of SCUs, a manual step for which only the reference summaries are used as input (the system summaries are not looked at yet). SCUs are defined as minimal meaning units in the text. The main idea is that many of these will be shared between summaries, which is the basis of assigning pyramid weights. There are also singleton SCUs that occur only in one reference summary. The annotator has to label each SCU by a short sentence in natural language to state the shared meaning. The weight of an SCU is the number of reference summaries it occurs in (if there are 4 reference summaries, a possible weight is between 1 and 4). It is expected that only a few SCUs which reflect the consensus of the reference summaries have large weights, but there will be many SCUs of small weights (as reference summaries differ in what details they choose to include). Hence, one can imagine a pyramid in which tiers from the bottom to the top contain SCUs of increasing weights.

The second step is the identification of SCUs in the system summaries, another manual step. The raw score of a system summary is the sum of the weights of the SCUs it contains. The optimal summary that is theoretically possible is one that picks up SCUs top-down tier by tier as length permits. The (normalized) score of a summary is defined as the ratio between the raw score of itself and the raw score of the optimal summary.

There are two possible definitions of the normalizing denominator, i.e. the raw score of the optimal summary. One definition is the maximum total weight that is attainable by the average number of SCUs of the reference summaries. The other definition is the maximum total weight that is attainable by the number of SCUs in the system summary in question, including additional SCUs that are not found

in the reference summaries. I use the first definition, because of two reasons: First, determining the number of singleton SCUs, especially in a machine-generated text, is difficult and has to involve a certain degree of arbitrariness. Second, because in my experiment, all summaries (both reference summaries and system summaries) are already of the same length, the normalizing denominator should be a constant across different systems. Packing as many SCUs as possible within the same word limit is also a part of the summarization ability, therefore a system should not be encouraged to output fewer SCUs to decrease the denominator.

There are attempts to semi-automate the pyramid method. Passonneau et al. (2013) score a summary by using distributional semantics to approximately match the summary against the SCUs in a predefined pyramid. Bauer and Teufel (2015) present a method to automatically score timeline summaries, in which they manually connect SCUs from the first step to events in the text. The events themselves are detected automatically by the TIPSem-B extraction system (Llorens et al., 2010). It is a compromise because it can only evaluate summaries that are created by choosing from a fixed number of events. These semi-automatic methods eliminate the need of labour in the second step, therefore an infinite number of system summaries can be analysed for free. This property is potentially useful in training a system that requires continual feedback on its summary quality. However, in an evaluation of existing summary outputs, the second step takes much less effort to perform than the first step. In addition, Louis and Nenkova (2009) present a fully automatic non-pyramid method, which does not require reference summaries. However, its correlation with human judgement is lower than that of ROUGE (which of course uses reference summaries).

The ideas behind propositions and SCUs are similar, but there are important differences. One difference is that the size of an SCU is often larger than a proposition. Some SCUs, such as one labelled "*Two Libyans were indicted*", can be conveyed by a single proposition. But more than one proposition is necessary to cover the SCU "*Wales is divided on issues of separation*", for example:

```
divide (Wales, on:  issue)
of (issue, separation)
```

This is further complicated by variations in the surface forms: In the sentence "*In Wales, countrymen are divided on issues of separation*", the same SCU maps

to three propositions instead of two. This points to the other difference, namely that SCUs are created using human intelligence, whereas the propositions in my summarizer are created automatically. The automatic proposition building process, no matter by which currently available method, cannot be completely free from limitations on recognizing meaning from linguistic expressions. That is to say, automatic proposition building is not robust enough to support fair pyramid evaluation (at least, no such study exists). I therefore use the pyramid method in its conventional, i.e. text-based and manual, way.

5.2 Corpus of texts and summaries

For both ROUGE and the pyramid method, a corpus of texts and gold standard summaries is required. I have created such a corpus for the experiments in this chapter, by choosing to use educational texts and having fixed-length summaries written by human subjects.

5.2.1 Choice of text

Previously, news texts were commonly employed for the evaluation of summarization systems, for example in the Document Understanding Conference (DUC)[2] 2001–2004 tasks. However, news texts have a few properties that complicate the evaluation of a proposition-based summarizer:

First, news texts are typically written in the journalistic style, which calls for an abstract-like lead. This means the importance of text pieces can be guessed from their location in the document without actual understanding. Traditionally, many summarizers struggled to beat the lead baseline (Lin and Hovy, 2003), which is a "summary" created by extracting the first n words of the input. Of course, lead summaries can be used to train summarizers, but to evaluate intelligent summarizers, the texts must not be susceptible to such simple exploitation.

Second, news reports often presuppose a lot of external knowledge, including expert knowledge in an area such as economics or US politics, as well as contextual knowledge of related news in the same period of time. With the modelling of knowledge being a major difficulty in NLP, the essence of simulating comprehension would be unnecessarily obfuscated by the knowledge bottleneck. This is not to

[2] `http://www-nlpir.nist.gov/projects/duc/guidelines.html`

say that toy texts must be used, but rather we should use naturally-occurring texts that can be mostly understood using only lexical knowledge and knowledge given in the text itself.

In addition, there are issues with specific datasets which make them unsuitable for my experiments: DUC 2003 and 2004 only have multi-document summaries and very short ($\leq$ 75 bytes) single-document summaries; DUC 2002 have single-document summaries but most documents have only one associated summary; in DUC 2001 the reference summaries were created by sentence extraction, which cannot accurately reflect the most desired information.

I therefore introduce new evaluation materials, created from the *Official IELTS Practice Materials*.[3] The IELTS (International English Language Testing System) is a standardized test of English proficiency for non-native speakers, and it has two tracks: an Academic test and a General Training test. I use the reading sections of Academic tests, because they contain long texts selected from actual publications including books, journals, magazines, and newspapers. The texts cover various topics, some written in a descriptive way and some with argumentative features, and resemble popular science or educational articles. They are carefully chosen to be of the same difficulty level, and understandable by a non-specialist audience. Compared to news texts, both their content and their style are more general, thus they are suitable for demonstrating the domain-independent processing of propositions. Compared to the alternative of narratives or short stories, they have the advantage of representing a broad range of entities and types of relations instead of focusing on the life events of a particular protagonist. They also contain fewer occurrences of direct speech compared to news and stories, making it less challenging to establish coherence among propositions.

5.2.2 Elicitation of summaries

Out of all 108 texts of IELTS volumes 1–9 (4 tests per volume, 3 texts per test), I randomly sampled 31 for my experiments. I then elicited 4 summaries for each, written by 14 native or highly proficient speakers of English, i.e., a total of 124 summaries.[4] Most human subjects are members of the University of Cambridge.

[3] At the time when the data was collected, nine volumes in this series had been published. For example, the ninth volume is *Cambridge IELTS 9: authentic examination papers from Cambridge ESOL*, Cambridge University Press, ISBN: 9781107615502.

[4] The summaries are available for download at `http://www.cl.cam.ac.uk/~yf261`.

The maximum number of summaries per person is 31, the minimum number is 2.

In the guideline, the human subjects were asked to create natural-sounding text, keeping the length strictly to 100 ± 2 words (to facilitate this, they were recommended to first write freely, and then add or subtract information in a text editor that displays word count). The guideline given to the human subjects also specifies some principles on the quality of summaries, as shown in the following excerpt:

> A summary should be –
>
> **A paragraph in natural English.** Please write complete and coherent sentences and do *not* enumerate pieces of information in bulleted lists. You may write sentences of variable lengths and structures, and organize and synthesize information as you would normally do.
>
> **Informative of the original.** Please do *not* deliberately substitute words or change spelling, but overall just write the best summary you can produce, which is natural to you and reasonably close to the text. Extracting sentences is generally *not* an effective utilization of your 100 words. Do *not* add meta-information like "this text is about X and discusses several problems that X can cause"; instead, say "X can cause problems A, B, and C".
>
> **For a general reader.** *Do* include something from the original text that you consider boring but a general reader needs to know in order to understand the text. On the other hand, do *not* introduce your own knowledge or conclusion if it is not given directly in the text.

The requirements of controlled length and no excessive paraphrasing are due to the limitations I mentioned earlier, i.e. ROUGE's sensitivity to length variations, and the lack of paraphrasing detection. Because the human subjects are asked to write a complete and coherence paragraph, considering the amount of information to be covered, the amount of simple extraction is nonetheless expected to be low. Additionally, as a protective measure against copy-and-paste, the source texts were provided in printed copies or PDF files in which texts are converted into vector graphics.

5.2.3 Characteristics of the texts and summaries

The average length of the selected source texts is 832 words (standard deviation: 107 words). For 100-word summaries, the compression ratio of the summaries is approximately 8 : 1. To measure the readability of the texts, I use the "SpaCy Readability" package[5] to determine the Flesch–Kincaid grade level (Kincaid et al., 1975), which is based the average number of words per sentence and the average number of syllables per word. The average Flesch–Kincaid grade level of the texts is 12.5 (standard deviation: 1.9), corresponding to a level between high school (grades 10–12) and undergraduate education. This finding is consistent with the properties of IELTS reading texts, i.e. entry-level academic content or popular science. Under this measure, the texts' readability is similar to that of news, but dependence on external knowledge is not directly incorporated in the formula. In conclusion, the texts are of suitable difficulty for the evaluation of summarization systems or document understanding systems in general.

A small number of the texts in the published books are divided into sections with section headings. However, considering that none of the experiment systems can make use of this information, the headings have been removed from the texts (for both systems and humans).

Depending on how much copying the human summarizers used, their summaries will be more or less abstractive. I tried to quantify this by computing the (case-insensitive) unigram, bigram, and trigram overlap between each summary and its source text. The unigram overlap is defined as a fraction $\frac{|x \cap y|}{|x|}$, where x is the set of unigrams in the summary, and y is the set of unigrams in the source text. Bigram overlap and trigram overlap are similarly defined (tokens in a bigram or trigram do not cross sentence boundaries). The average unigram, bigram, and trigram overlap are 79%, 42%, and 25%, respectively (if summaries were fully extractive, these numbers would have been 100%). The fact that overlap decreases drastically from unigram to trigram means the summaries are highly abstractive. The relevant statistics of individual texts are listed in Appendix C.

[5] `https://github.com/mholtzscher/spacy_readability`

5.3 Extractive system

In this section, I will evaluate the content selection ability of my summarizer by extractive generation. I will first describe how textual summaries are created based on the result of the simulation of comprehension, in Subsection 5.3.1. I will then present two experiments using this system. The first experiment evaluates different approaches of modelling argument overlap (Subsection 5.3.2). In the second experiment, my method of summarization by comprehension simulation is compared to other methods of summarization, including summarization that is not based on propositions as well as summarization in which propositions are used in a different way (Subsection 5.3.3).

5.3.1 Generation method

A simple way to realize a selected proposition is to output the original sentence which gave rise to it. This is a robust method because a large proportion of the meaning of a sentence can be understood on its own (with the notable exception of anaphoric expressions), thus ensuring truthfulness. Grammaticality of an extracted sentence is of the same level as the original text. The drawback, however, is that a sentence may host other propositions which are not selected but are nonetheless forced to be present in the summary. This is not as bad as it appears, because propositions which originate from the same sentence are usually highly correlated (for example in the form of embedded propositions) and depend on each other for interpretation.

Although sentence extraction makes my summarization system superficially similar to other sentence extractors, the two types of systems are clearly distinct in the reason why a sentence is extracted. In my system, a sentence is seen as merely a ready-made container of propositions, which in principle could be replaced by better containers to achieve a more compact representation of the same information. The processing and selection of information is separate from the representation of information. For example, as a future improvement, it may be possible to unify identical propositions from different source sentences. In that scenario, which sentence is extracted to realize a unified proposition would only depend on the need of a discourse model, i.e. which sentence makes a coherent and concise summary.

To generate a W-word summary (in the following experiments $W = 100$), I select sentences in the following way: I define the value of a proposition as the number of memory cycles it has been activated in; the value of a sentence as the total value of the propositions it contains. I want to maximize the total value of selected sentences, within the constraint of word count. In combinatorial optimization, this is a 0-1 knapsack problem: Given a set of items (sentences), each with a weight (the number of words in the sentence) and a value, find a subset of items to include in a knapsack (summary) so that the total weight does not exceed a given limit (W) and the total value is as large as possible. It is a special case of the bounded knapsack problem, in which the number of copies of each item to be included is not 0 or 1, but is bounded by some constant. For this task, it is clearly useless to include multiple copies of the same sentence in a summary.[6]

Let us denote the selection of sentence i as a binary variable $x_i \in \{0, 1\}$, the sentence value as v_i and the sentence length (in number of words) as w_i. Selecting sentences to output in the summary is formulated as the following problem:

$$\text{maximize} \qquad \sum_i v_i x_i \tag{5.1}$$

$$\text{subject to} \qquad \sum_i w_i x_i \approx W \tag{5.2}$$

Here, the constraint is expressed as summary length being approximately equal to W, instead of being less than or equal to W. The meaning of being approximately equal is as follows: I first obtain two solutions, one of maximal length $\leq W$, and the other of minimal length $\geq W$. If the length of either solution is within the range $[W - \epsilon, W + \epsilon]$ (in this case $\epsilon = 2$, the same range as the human subjects were told to write summaries in), the solution whose length is closer to W is accepted. Otherwise, I choose the longer solution and trim it to exactly W words (the sentence that has the lowest v_i/w_i ratio is truncated first). Because w_i are integers, this problem is solved using dynamic programming in $O(nW)$ time

[6] An alternative way to select sentences, which is similar in principle to SumBasic (Subsection 2.1.1), is to iteratively perform two steps: selecting the sentence of the highest value, and reducing the value of other sentences with which it has token (or other kind of) overlap. This strategy may possibly reduce repetition in the output summary by applying Maximal Marginal Relevance. I do not use it here because the result would depend on not only the simulation of comprehension, but also the algorithm controlling repetition.

(where n is the number of source sentences).

5.3.2 Testing models of argument overlap

I use the first experiment to determine which method, among the methods I have described in Chapter 4, performs the best in modelling argument overlap. There are two types of argument overlap models, i.e. the lexical chain (LC) model and the various vector space models. Additionally, I have created a dummy model, which assigns an overlap strength of 1 to pairs of arguments whose surface strings match, and 0 otherwise. Note that all these models are used in conjunction with coreference resolution, which serves as the first line of argument overlap detection.

The LC model has four variants (LC-0, 1, 2, 3), which are parameterized differently as defined in Section 4.3. The vector space models are prepared in the following way:

LSA I trained an LSA model using the English Wikipedia (2016) corpus. Articles shorter than 100 words, and words that too frequent (occurring in more than 20% of articles) or too infrequent (less than 20 occurrences overall) are excluded. During training, multi-word expressions are recognized using the algorithm I described in Section 4.2, trained on the same corpus. During overlap detection, multi-word expressions are looked up in the same way as this is done in the LC model, i.e. first trying to match the complete phrase and then falling back to shorter ones.

Word2vec I obtained the word vectors trained by the authors on part of Google News.[7] Due to the characteristics of its dictionary, I have to transform the text of a word or phrase in two ways before looking up: splitting a hyphenated word into individual tokens, and converting British spelling to American spelling. Multi-word expressions, including the results of splitting hyphenated words, are looked up in the same way as in the LSA model.

GloVe I obtained the word vectors trained by the authors on Wikipedia (2014) and Gigaword (fifth edition).[8]

[7] `https://code.google.com/archive/p/word2vec/`
[8] `https://nlp.stanford.edu/projects/glove/`

Model	Model properties				ROUGE scores			
	Corpus	#Dim.	MWE	Case	1	2	L	SU4
LC-0	/	/	+	+	0.369	0.110	0.335	0.147
LC-1	/	/	+	+	**0.376**	**0.119**	**0.343**	**0.153**
LC-2	/	/	+	+	0.373	0.115	0.339	0.149
LC-3	/	/	+	+	0.370	0.112	0.336	0.147
LSA	799M	300	+	+	0.356	0.095	0.326	0.134
Word2vec	100G	300	+	−	0.362	0.105	0.334	0.139
GloVe	6G	300	−	+	0.351	0.098	0.321	0.133
NMT	348M	620	−	−	0.356	0.097	0.324	0.136
Dummy	/	/	+	+	0.356	0.106	0.325	0.139

Table 5.1: Properties and summarization performance of argument overlap models. Among model properties, "corpus" refers to the number of tokens in the training corpus, "#Dim." is the number of dimensions of each word vector, "MWE" indicates whether multi-word expressions are modelled, and "case" indicates whether letter case is ignored by the model.

Neural machine translation (NMT) I obtained the word vectors from the authors. These word vectors are exported from their neural machine translation systems trained on English–French or English–German parallel data.[9]

5.3.2.1 Results and discussion

The performance of my summarization system using these models of argument overlap is reported in Table 5.1. On all ROUGE measures, LC-1 has the best performance. Among the LC variants, the fact that LC-1 is better than LC-3, which does not involve verbs, shows that the introduction of verbs into the lexical chains is advantageous. In the following discussion, I will focus on LC-1 as the representative of the LC model.

All LC variants are superior than all four vector space models, none of which outperforms the dummy model of overlap on all ROUGE measures (although word2vec has advantage on two out of four measures). Under the paired Wilcoxon test, which is a non-parametric version of the t-test for matched pairs, the difference between LC-1 and any non-LC model except word2vec is statistically significant ($p < 0.05$) on most ROUGE measures, while there is no significant difference among the non-LC models on any ROUGE measure (except that LSA is significantly

[9] `https://www.cl.cam.ac.uk/~fh295/`. Both English–French and English–German models are available for download, but the vectors turn out to be identical. In Table 5.1, the size of training corpus of the English–French model is reported.

worse than the dummy model on ROUGE-2). This shows that lexical chains model argument overlap effectively.

This result is different from my previous work (Fang and Teufel, 2016) in that the statistical significance of the difference between LC-1 and word2vec is lost, which is mainly due to the improved performance of word2vec. This improvement can be attributed to the lexical transformations I added in this version of my summarizer to handle hyphenation and spelling differences. Of course, performance may be influenced by other factors such as changes in the proposition building module, but the interaction between these factors and argument overlap is relatively remote. Considering there is no significant difference between word2vec and other vector space models or the dummy model, LC is still currently superior to word2vec in modelling argument overlap. In the subsequent experiments, the LC model will be used.

The success of LC does not mean that argument overlap cannot be possibly modelled by data-driven approaches. This is evident from the fact the dummy model performs within the range of the vector space models, while it is obvious that these vector space models are more informative about word meaning than the dummy model. It is likely, as I have motivated in Chapter 4, that argument overlap mostly requires near-synonyms, and other types of word associations should be given much smaller weights. In other words, argument overlap is sensitive to noise (spurious overlaps).

To test the hypothesis that vector space models are suffering from the numerical problem of insufficient contrast in the cosine similarities of word vectors, I have experimented with a few non-linear transformations to the similarity scores. One type of transformation is the power function $\hat{x} = x^k$, where $x \in [0, 1]$ and $\hat{x} \in [0, 1]$ are the similarity scores before and after transformation, and $k > 1$ is a constant (I tested $k = 2$ and $k = 3$). As a convex function, it increases the difference between two high similarity scores, making argument overlap more selective. An other transformation I experimented with is the function $\hat{x} = \frac{1-\cos(x\pi)}{2}$, which has an S-shaped curve in $[0, 1]$ and therefore magnifies the difference between two similarity scores close to 0.5. This function can be applied recursively to increase the polarizing effect (I tested applying once and twice). However, when used together with every vector space model, none of these transformations lead to performance gains on all four ROUGE measures. Hence, I must conclude that it is inherently difficult to distinguish strong and weak overlaps using the current

method as I have described.

One of the potential strengths of vector space models is in modelling compositionality. I have experimented with two possible ways to use vector composition (vector mean is used as the composition function). In one experiment, I compose vectors of individual tokens to represent a multi-word expression. For models that already contain multi-word expressions, the vectors to be composed can also correspond to subsequences of an expression. In the other experiment, I obtain a vector representation of every proposition by composing the vectors of its functor and arguments (which can be embedded propositions). Thus, it is possible to calculate the overlap between an argument that is a word and another argument that is a proposition. However, the advantage of vector compositionality is not demonstrated in these experiments. Of course, compositional distributional semantics is an active area of ongoing research; it is possible that the problems we face now (such as the lack of a way to model instances) will eventually be overcome.

5.3.3 Testing methods of summarization

In the second experiment, I compare the performance of my summarizer to four non-proposition-based summarizers: MEAD, LexRank, TextRank, and SummaRuNNer (Nallapati et al., 2017), which I have introduced in Chapter 2, as well as the lead baseline. I use out-of-box implementations of MEAD, LexRank, and TextRank.[10] In addition, I also compare to the proposition-based Graph baseline, which is defined in Section 3.4.

SummaRuNNer is a neural network-based extractive summarizer. As I have described in Subsection 2.2.4, it uses recurrent neural networks (RNNs) to first combine the embeddings of the words in each sentence into a sentence vector, and then combine all sentence vectors into a document vector. Its open source implementation[11] contains two variant neural network architectures in addition to the original model. One of them, called "CNN–RNN", uses convolutional neural networks (CNNs) instead of RNNs to compute sentence vectors from word embeddings, but still uses an RNN to combine the sentence vectors together.

[10]For MEAD, the official implementation is obtained from `http://www.summarization.com/mead/`. For LexRank and TextRank, the implementation in the open source library Sumy (`https://github.com/miso-belica/sumy`) is used.

[11]`https://github.com/hpzhao/SummaRuNNer`

The other variant, called "hierarchical attention networks", adds the attention mechanism to the RNNs on both levels (word-level and sentence-level). I have run all three models, but I report only the best performing model, which was CNN–RNN. The models used here are trained on the CNN / Daily Mail corpus, which is also the training corpus the authors used. Because it was impossible to train the deep learning models on the same amount of IELTS material, the performance seen here is an underestimate of these models. (On the other hand, my system is unsupervised.)

For the sake of fairness (considering that the ROUGE F-score penalizes summaries that are shorter or longer than the gold standard summaries), I have changed the implementation of SummaRuNNer so that, instead of outputting the top k sentences (where k is the user's choice), a summary of length 100 ± 2 words is output. This is achieved by selecting sentences in descending order of their output probabilities until reaching at least 98 words, and if longer than 102 words, truncating the least probable sentence at the end to make the output exactly 100 words.

I also enforce the same output length in the other systems, except that I do it in a different way for MEAD, in which the desired summary length is specified as the maximum number of words instead of number of sentences. For this system, I first request a 100-word summary, and if the output is too short, I gradually increase the word limit and truncate if necessary.

5.3.3.1 Results and discussion

The ROUGE results of the systems on the IELTS corpus are reported in Table 5.2.[12] On all four ROUGE measures, my summarizer outperforms all other systems, and is the only system that beats the lead baseline on all measures (most other systems only manage to beat it on ROUGE-1). As before, I use the paired Wilcoxon test to validate the differences statistically. On all four ROUGE measures, the difference between my system and each other system is statistically significant ($p < 0.05$), whereas the difference between any two non-proposition-based systems (including the lead baseline) is insignificant ($p > 0.05$).

The best performing Graph summarizer is Graph-B, which uses betweenness

[12]The ROUGE-1, 2, L, SU4 scores of the previous version of my system are 0.376, 0.122, 0.345, 0.154, respectively (Fang and Teufel, 2016).

System	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-SU4
Mine	**0.376**	**0.119**	**0.343**	**0.153**
Graph-B	0.353	0.092	0.320	0.132
Graph-CD	0.344	0.084	0.311	0.127
Graph-CU	0.344	0.086	0.313	0.127
Graph-ED	0.337	0.083	0.306	0.124
Graph-EU	0.331	0.075	0.300	0.118
MEAD	0.343	0.092	0.308	0.128
LexRank	0.349	0.087	0.316	0.129
TextRank	0.343	0.094	0.309	0.130
SummaRuNNer	0.348	0.097	0.320	0.130
Lead	0.341	0.100	0.314	0.132

Table 5.2: ROUGE scores of summarization systems

centrality. No statistically significant difference among the Graph variants or between the Graph variants and the non-proposition-based systems is detected under the paired Wilcoxon test (except that Graph-EU is significantly worse than LexRank, Graph-B, Graph-CU, and Graph-ED on one or two ROUGE measures). The fact that the Graph summarizer is in the same ballpark as the non-proposition-based summarizers and is significantly worse than my system indicates that the main advantage of my system originates from the simulation of comprehension. Although the simulation is a highly simplified realization of KvD's model of human text comprehension, it does capture at least some common features in systems which attempt to model an intelligent comprehension process. For instance, by modelling the working memory as a tree, only the strongest connections are considered whereas alternative interpretations are discarded, which is similar to L^1-normalization in machine learning in their effect of enforcing a simple explanation with sparse values. The softmax function, which is commonly used in neural network models, can also be regarded as a kind of competition in which all options except the one with the highest score are suppressed because of very low probabilities. Furthermore, the incremental processing entails that new information is understood on the basis of old information, which is difficult to model using one static graph but is common in recurrent neural networks and reinforcement learning.

5.4 Abstractive system

Abstraction, rather than extraction, is generally seen as the more desirable method of summarization. In its original sense, abstraction means processing and selecting information on a meaning representation that is independent from the surface forms in the original text. However, it can also refer to the ability of re-generating any expression that is not found in the original text. My summarizer, by virtue of being based on propositions, has an abstractive processing model, but extractive generation has been used in the previous experiments. On the other hand, there are summarizers which do very little abstraction in processing, but use abstractive natural language (NL) generation techniques to achieve a better presentation of the same information of the original sentences. In this section, I will first show how abstractive generation is incorporated into my summarizer (Subsection 5.4.1), and then compare the resulting summarizer to summarizers which apply abstractive generation on non-propositions.

Due to the difficulty in finding a meaning representation for general text as well as a processing model that works on that representation, most existing works on abstractive summarization concern compressing individual sentences or merging sentences of similar content. These works usually assume a pipeline model of summarization in which sentence extraction is performed independently before sentence compression or sentence fusion. The only exceptions are Martins and Smith's (2009) system, and Nishikawa et al.'s (2014) system for Japanese text, both of which optimize sentence selection jointly with sentence compression. Therefore, I have created four pipeline systems for comparison (Subsection 5.4.2), two of which use my proposition-based summarizer as their sentence extractor, and the other two depend on LexRank, one of the best performing competitors in extraction, for sentence extraction. Finally, I will show using experiments that abstractive generation from propositions produces summaries of higher quality than those produced by the same abstractive generation via the middle-man of sentence extraction (Subsection 5.4.3).

A different but promising method, which also has an abstractive processing model, is end-to-end neural networks. Such a system does not model propositions or the memory cycles explicitly, but the large number of trainable parameters in it are thought to implicitly hold some rules useful for document understanding. Therefore, I also compare to a state-of-the-art end-to-end system (Subsection 5.4.2).

5.4.1 Generation method

In the following experiments, the ACE processor[13] is used for NL generation from propositions. ACE is a software system developed as part of the DELPH-IN initiative[14] and LinGO project[15]. It is designed to process DELPH-IN HPSG wide-coverage, linguistically motivated grammars such as the English Resource Grammar (ERG) (Flickinger, 2000; Flickinger et al., 2014), a broad-coverage, symbolic, bidirectional (i.e. supporting both parsing and generation) grammar of English.

The ERG uses Minimal Recursion Semantics (Copestake et al., 2005, MRS) as its semantic representation. My co-author Haoyue Zhu has devised a method to match the basic units of MRS, which are called elementary predications or EPs, with my propositions. This is of course sub-optimal, because two parsers, one syntactic and one semantic, have to be run on one document, and the resulting two parses may disagree. During the write-up of this thesis, a new collaborative project of creating an ERG-based proposition building module has been initiated, which will eventually replace the temporary measure of having to align two parses.

The MRS structure of a sentence can be represented in the form of a dependency graph called Dependency MRS or DMRS (Copestake, 2009), in which every node represents an EP. The way Zhu selects EPs based on the ranked list of summary propositions (result of the memory cycles) can be summarized as the following steps on the DMRS graphs of the input sentences: The summary propositions are realized one by one, each with reference to the DMRS graph of its corresponding input sentence; if another proposition which originates from the same input sentence is encountered, the algorithm is rerun on that sentence taking into account all propositions processed so far.

For each proposition, an initial set of nodes is selected based on the textual tokens contained in that proposition. Then the node set is expanded for both grammaticality (ensuring arguments of a selected EP are also selected) and graph connectivity (which is required for successful generation). Several strategies are used to ensure grammaticality, including recovering prepositional complements of verbs, recovering quantifiers, and dismantling sentential coordination. The strategy for graph connectivity is to iteratively grow the currently largest connected

[13]The Answer Constraint Engine, `http://sweaglesw.org/linguistics/ace/`.
[14]Deep Linguistic Processing with HPSG, `http://www.delph-in.net`.
[15]Linguistic Grammars Online, `http://lingo.stanford.edu`.

component, adding one node at a time.

Nodes in a DMRS graph can also be modified. Currently, this is used to generate "*there be*" sentences when a proposition is expressed as a noun phrase in the input text, but in principle more intelligent solutions (such as denominalization) are possible. Technical details, such as the handling tokenization differences, can be found in our paper (Fang et al., 2016).

To evaluate the effect of the ERG-based generation on both content selection and text quality, I have created a baseline NL generator which simply outputs word tokens from the summary propositions. In this token-based extraction, I iterate through the propositions in descending order of their score, and put the tokens of the proposition into the set of summary tokens, until the word limit is reached. The tokens of a proposition include content words and functional words, but embedded propositions are only represented by their functors (rather than recursively extracting their tokens). Finally, the summary tokens are sorted by their order in the source text, and sentence breaks are added. For human readability, punctuations are recovered using heuristics, although their existence has no influence on ROUGE evaluation.

The intention of token-based extraction is to faithfully represent the selected content, with no guarantee on truthfulness or grammaticality. Considering the imperfections in the parsing and proposition building process, and the fact that embedded propositions are not recursively generated, there is a high chance that some obligatory arguments are missing. For example, let us consider the following source sentence:

> *In Britain, for example, the dull weather of winter drastically cuts down the amount of sunlight that is experienced which strongly affects some people.*

The tokens which correspond to the propositions selected by the summarizer are underlined. The "sentence" produced by token-based extraction, which only contains these tokens, would be uninformative and ungrammatical:

> *In Britain, the weather cuts down the amount strongly affects.*

In contrast, the ERG-based generation is much better:[16]

[16]Note that the adverbial modifier "*strongly*" appears in a different place from the source sentence, but is still grammatical. This is an effect of the generator's degree of freedom when verbalizing a semantic representation.

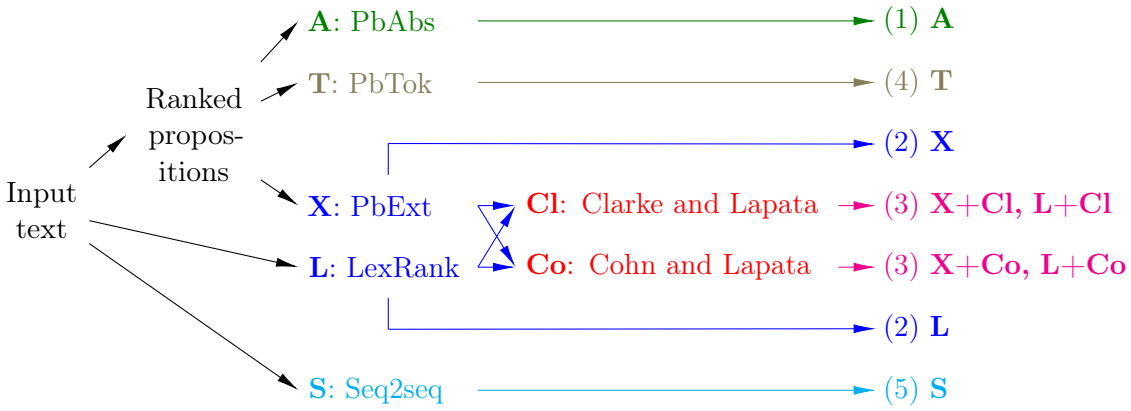


Figure 5.1: The organization of different summarization systems

> *In Britain, the weather cuts down the amount of sunlight, which affects some people, strongly.*

In Subsection 5.4.3, the ERG-based generation and the token-based extraction are not only compared in terms of ROUGE scores on content selection, but also by human evaluation on text quality.

5.4.2 Systems

In the following experiments, there are in total nine summarization systems (Figure 5.1), which can be classified into five categories:

1. Proposition-based incremental processing with abstractive generation (PbAbs);[17]
2. Sentence extraction, either using on a proposition-based incremental processing model (PbExt) or a shallower content selection model (LexRank);
3. Sentence extraction followed by sentence compression ({PbExt, LexRank} + {Clarke and Lapata, Cohn and Lapata});
4. Proposition-based incremental processing with token extraction to represent propositions (PbTok);

[17] When the ACE processor fails to parse a sentence, the summary propositions in that sentence are realized using PbTok. The coverage of the ACE processor over all 108 IELTS documents is 86.5%.

5. Sequence-to-sequence abstractive summarization (Seq2seq).

All proposition-based systems (PbAbs, PbTok, PbExt) use lexical chains for argument overlap, which is found to be the most effective method in the previous section. I choose LexRank as the alternative content selection model because it is the best performing non-proposition-based competitor in the extractive experiment presented in the previous section.[18] The purpose of using an alternative is to exclude the possibility that PbExt is unsuitable for post-processing (sentence compression) even though it outperforms all other systems when used alone.

The generation methods of category 1 and 4 have been described in the previous subsection, whereas category 2 is taken unchanged from the previous section. I will now describe how the sentence compressors are used in the systems of category 3. After that, I will turn to the seq2seq system representing category 5.

5.4.2.1 Sentence compressors

Sentence compression using machine learning and/or constrained optimization is an active area of research. A highly influential model is the noisy-channel model (Knight and Marcu, 2000, 2002), which formulates the problem as finding the compressed sentence s for the original sentence l which maximizes $P(s)P(l|s)$. Common methods use syntactic, lexical and discourse-based features to determine the words to be dropped or paraphrased (McDonald, 2006; Clarke and Lapata, 2008; Cohn and Lapata, 2007, 2008; Yoshikawa et al., 2012). More recently, neural language models have also been applied to the problem (Rush et al., 2015).

One sentence compressor I use is the work of Clarke and Lapata (2008), who use integer linear programming (ILP) to find the optimal compression of a sentence within linguistic constraints. In this model, compression is realized in the form of word deletion. I use its unsupervised version,[19] which only requires a corpus to induce the language model and the significance scoring function. The 100M-word British National Corpus (BNC) is used for this purpose, on which I trained a trigram language model with interpolated Kneser-Ney smoothing and the "unknown word" token enabled.

[18] SummaRuNNer, which also performs competitively, was published one year after this experiment was done and published. It was later added to the thesis to reflect the latest development on extractive summarization.

[19] The implementation is obtained from `http://github.com/cnap/sentence-compression`.

The other sentence compressor I use is created by Cohn and Lapata (2007).[20] It is a supervised tree-to-tree transduction method, which can be regarded as a variant of Knight and Marcu's (2002) model based on synchronous context-free grammar. But unlike Knight and Marcu's model, their model is capable of tree-rewriting operations beyond subtree deletion. In addition to a language model (for which the BNC language model is used again), this system also requires parallel data, i.e. pairs of source and compressed sentences and their parse trees (which I obtain using the Stanford parser). Instead of the Broadcast News Corpus they used, I train the tree-rewriting model on the full Written News Corpus, which only became available later (Clarke and Lapata, 2008). Following their example, I align the words between the source sentences and the compressed sentences using GIZA++ (Och et al., 1999), using not only the pairs of training sentences but also additional pairs of single words (so that every word of the lexicon can be aligned with itself). The alignment is symmetrized using the intersection heuristic (Koehn et al., 2003).

In a pipeline summarization system, a sentence compressor is first run on every sentence of the input document. Then, a summarizer (PbExt or LexRank) is run on the original (uncompressed) text, but when it extracts sentences, the compressed sentences are output in place of the original ones, which means more sentences can be extracted than not using compression, given the same word limit.[21]

5.4.2.2 See et al.'s (2017) system

The abstractive summarizer by See et al. (2017),[22] which is introduced in Subsection 2.2.4, is used here as the representative of end-to-end (i.e. non-pipeline) systems. It operates a sequence-to-sequence model, with a pointer–generator network, the attention mechanism, and the coverage mechanism. Similar to SummaRuNNer in the previous section, it is trained on the CNN / Daily Mail corpus of texts and summaries. Its power derives from the fact that it is strongly supervised using a large amount of data, as well as from its coverage mechanism,

[20] The implementation is obtained from the authors.

[21] In my experiments, the Cohn and Lapata compressor fails to process 8 input sentences. If any of them is selected for summary, the uncompressed sentence will be output.

[22] The implementation is obtained from the authors at `https://github.com/abisee/pointer-generator`.

which explicitly penalizes redundancy.

In order to control the output length for a fair comparison, I have modified its implementation of the beam search decoder. Originally, the decoder is controlled by the maximum and minimum number of decoding steps, each step corresponding to the output of one token. Punctuation marks are counted towards decoding steps, but should not contribute to the length of a summary (in number of words). This often produced summaries which were too short. To rectify this problem, I changed the decoding process so that non-word tokens are identified, and the most probable summary is selected from candidate sequences that are 100 ± 2 words long. This leads to noticeably improved results over the original decoder.

It is also worth noting that the implementation by default disables the coverage mechanism, and limits the document encoder to 400 steps, probably for efficiency reasons. To give the model the best possible chance, I enable the coverage mechanism and set the maximum number of encoding steps to 1200 (which is more than enough for texts in my corpus). I have verified that the new setting leads to better results (a 12% improvement over the default setting and the old decoder on ROUGE-1 on my dataset).

5.4.3 Experiments

As before, the abstractive system is evaluated using the IELTS corpus. The first experiment is a ROUGE-based evaluation. In addition, two human experiments have been done, among which one evaluates content selection using the pyramid method, and the other is quality judgements of language generation.

5.4.3.1 ROUGE evaluation

The ROUGE scores of all nine systems are reported in Table 5.3. As before, statistical significance is computed using the paired Wilcoxon test. Although PbExt is the best system in all metrics, PbAbs performs comparably to it on ROUGE-1 and ROUGE-L. It performs at least as well as the four pipeline systems. This is an achievement because the goal of PbAbs is to faithfully reflect the summary propositions, and is therefore only directly comparable to PbTok. It is an improvement over PbTok, which is significantly worse than PbExt in all metrics except ROUGE-L.

System	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-SU4
1. Abstract generation from propositions				
PbAbs (A)	**0.364**	0.088	**0.340**	0.131
2. Sentence extraction				
PbExt (X)	**0.376**	**0.122**	**0.345**	**0.154**
LexRank (L)	0.349	0.087	0.316	0.129
3. Sentence extraction + compression				
X + Cl	0.361	**0.090**	0.335	**0.132**
X + Co	0.340	0.074	0.321	0.113
L + Cl	0.356	0.077	0.325	0.126
L + Co	0.336	0.067	0.314	0.110
4. Token extraction for propositions				
PbTok (T)	0.356	0.088	0.336	0.130
5. Non-proposition end-to-end abstract generation				
Seq2seq (S)	0.336	0.081	0.309	0.121

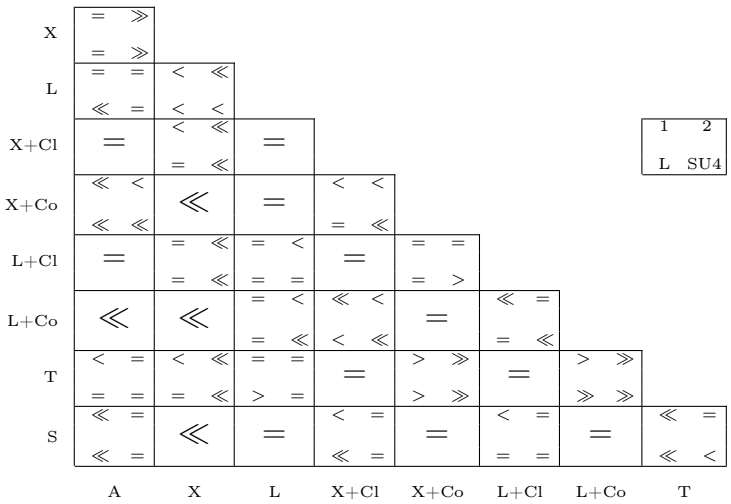


Table 5.3: ROUGE scores and statistical significance of the differences. The top two results of each column are printed in bold. The four positions of every cell of the significance table (as shown by the legend) correspond to ROUGE-1, 2, L and SU4, respectively. "$\gg$" means row statistically outperforms column at $p < 0.01$ significance level, "$>$" at $p < 0.05$ significance level, and "$=$" means no statistical difference detected.

ROUGE seems to penalize any act of sentence regeneration. This is more pronounced for ROUGE-2 and SU4, which are different from ROUGE-1 and L in that they target at only the bigrams which are close to each other in text. A typical example is that PbAbs is significantly inferior to PbExt on ROUGE-2 and SU4, but not on ROUGE-1 and L. A similar effect is observed between LexRank and its combination with sentence compressors (L+Cl, L+Co). Let us define compression rate, for both abstractive generation from a sentence and compression of a sentence, as the length of the generation or compression divided by the length of the source sentence. When compression rate decreases, the shorter sentences give rise to fewer n-grams, and those bigrams which are the target of ROUGE-2 and SU4 are easily lost due to the elimination of modifiers, etc. Consequently, any generation imperfection is disproportionately penalized by the numerical scores.

The extreme is seen in PbExt, which has the highest ROUGE scores but does not compress at all. A sentence compressor can benefit under ROUGE from being close to full-sentence extractions. This is the case for the Clarke and Lapata compressor, which only compresses to 67%. Therefore, it results are close to their uncompressed counterparts (e.g. X+Cl is similar to X).

In contrast, because the goal of PbAbs's generation module is not to output the generally important parts of a sentence, but to output specific information as dictated by the summary propositions, it faces a bigger challenge by sometimes having to perform extensive transformation on a long source sentence in order to present a small bit of information. Hence, it has a compression rate as low as 33%, i.e. it is heavily penalized by ROUGE. Nonetheless, it easily beats both systems using the Cohn and Lapata compressor, which compresses to 43%. This advantage, and the fact that it performs indistinguishably from X+Cl, show that PbAbs is highly efficient in terms of information packaging.

The sentence compressors have an effect of narrowing the performance gap between PbExt and LexRank, which is expected. On itself, PbExt significantly outperforms LexRank on every ROUGE metric. However, when they are combined with a sentence compressor, there is no longer significant difference between the results of X+Cl and L+Cl, or X+Co and L+Co. The reason for the diminishing of difference is that sentence compressors are also summarizers by themselves, which will condense any input they are given. Because the information of which parts of the sentences are important (or for what reason a sentence is selected) is not passed to the sentence compressors, a compressed sentence is only locally optimal

for the sentence itself. The implication, therefore, is that end-to-end systems are more sensitive to improvements than systems which require going through the middle-man of sentence extraction.

Finally, let us turn to the end-to-end competitor, Seq2seq. The advantage of PbAbs over Seq2seq is strongly significant ($p < 0.01$) on ROUGE-1 and L, but not significant on ROUGE-2 and SU4. Because the definition of compression rate is not applicable to Seq2seq, I use the notion of n-gram overlap with the source text, which I have defined and used in Subsection 5.2.3, to estimate its abstractiveness. The average unigram, bigram, and trigram overlap of the outputs of Seq2seq with the source texts is 99%, 98%, and 95%, respectively, which is much higher than PbAbs's 97%, 71%, and 53%. This means although Seq2seq has the ability to choose when to copy from the source text and when to generate new expressions, copying in large chunks is still the dominating behaviour. In fact, Seq2seq is more extractive than all four pipeline systems in terms of bigram and trigram overlap with the source text. Therefore, Seq2seq being close to PbAbs on ROUGE-2 and SU4 is a result of it being on the "safe" side of extraction. In the following evaluation, I will use the pyramid method to show that PbAbs is advantageous over Seq2seq in content selection.

5.4.3.2 Human evaluation for content selection

I use the pyramid method to compare the two fully abstractive systems: PbAbs and Seq2seq. I followed the pyramid annotation guideline of DUC 2006 and used the accompanying annotation tool.[23] In the annotation of system summaries, the connection between a clause and an SCU is sometimes vague, for example due to the omission of necessary parts of a sentence, or because the selected information does not fully entail the SCU. For these cases, half points are given. Note that it is not required that the clause is interpretable out of its context; as long as an SCU can be understood by reading the entire output, the full weight of the SCU is awarded.

I randomly sampled 6 out of the 31 documents used in the previous evaluations. For each document, I annotated the SCUs of the four reference summaries it has; in total, 24 reference summaries were annotated. I present the annotation of

[23]The instructions and software are obtained from `http://www1.cs.columbia.edu/~becky/DUC2006/2006-pyramid-guidelines.html`.

the reference summaries for one document as Table 5.4. In this table, SCUs are identified for each sentence of each reference summary. Each SCU is represented by an SCU number. The definitions of all reference SCUs are listed in Table 5.5, each consisting of an SCU number, a label, as well as a weight.

After obtaining the reference SCUs (and not changing them afterwards), I identified the appearance of these SCUs in the summaries generated by the two systems. The annotation of system summaries of the same example is presented as Table 5.6. A starred SCU number indicates a partial overlap between the sentence and the SCU. The output of PbAbs contains SCUs 2, 3, 4, 5, 15, and partial SCUs 13, 14, 23; their total weight is 19.5. The output of Seq2seq contains SCUs 2, 5, 9, 23, and a partial SCU 17; their total weight is 11. To determine the score of the summaries, we also need the raw score of the optimal summary. On average, each reference summary contains 14 SCUs. The maximum total weight that is achievable with 14 SCUs is 39. (The numbers 14 and 39 are automatically computed by the annotation software.) Therefore, the score of the PbAbs summary is $19.5/39 = 50\%$, and the score of the Seq2seq summary is $11/39 = 28\%$.

The result of the pyramid evaluation is provided in Table 5.7. The average score of PbAbs is 34%, as opposed to Seq2seq (22%).[24] The advantage of PbAbs over Seq2seq is not only because it retrieves more reference SCUs than Seq2seq, but also because it selects more important SCUs than Seq2seq, which often reproduces less-weighted details. This is evidenced by the fact that the average weight of the SCUs which PbAbs selects for a summary is 2.5, whereas that of Seq2seq is 2.0.

A weakness of PbAbs (compared to PbExt) is its tendency to omit modifiers. This has a negative impact especially when key information is conveyed in the form of an adjective or an adverb, as opposed to nouns or verbs. For example, it generated the following sentence:

> *All of the languages have been studied in the century approach.*

But to fully deliver the information, it should also have generated "*studied prescriptively*" and "*the 18th century*":[25]

> *All of the languages have been studied prescriptively in the 18th century approach.*

[24] Due to the small data size, the paired Wilcoxon test only reached marginal significance ($p = 0.18$).

[25] "*18th*" is classified by the Stanford parser as an adjectival modifier of "*century*"; the problem would not occur if it was classified as a noun compound modifier.

Summary Text	SCUs
Reference Summary 1	
The picture for building pyramids of Egypt might have been helped by kites.	2, 4, 15
To verify this idea, an experiment was conducted to verify the idea.	5
From the results, the kite was able to lift a 33.5 tone column clean off the ground.	3, 8
However it remains unclear that what the Egyptians actually did.	11
The experiments also left many specialists unconvinced.	6
One argued the lack of the evidence of ancient kites.	26
Others felt there were physical evidence that that Egyptians were interested in flying.	10
However, the experiments might have seen practical uses nowadays, such as putting up building roofs with kites.	1, 19
Reference Summary 2	
The pyramids were built 3000 years ago, but nobody knows how as there are no pictures showing the construction of the pyramids.	4, 13, 16, 25
The Egyptians may have used kites to lift the stones in place.	2, 15
Maureen Clemmons noticed a hieroglyph of men apparently using a kite.	17, 23
Aeronautics professor Gharib experiments using kites as heavy lifters.	5, 7, 14
A 33.5-tonne column was successfully lifted with a 40-sqm sail.	3, 8, 9
The pyramid builders could indeed have used kites.	2
Many specialists are unconvinced, but some agree.	6, 21
The ancient Egyptians were interested in flight.	10
Kites can be used today to construct buildings in places where heavy machinery cannot be used.	1, 12
Reference Summary 3	
How ancient people built the pyramids remains a mystery and it is suspected that kites were used.	2, 4, 13, 18
To show if this is possible, aeronautics professor Morteza Gharib set up experiments to carry stone with only wind.	5, 7, 14
He succeed and with a 40-square-meter rectangular nylon sail, the stone column was lifted, which showed the proof of concept that people can build pyramids with the aid of kites.	2, 3, 9
Despite the possibility, it is not well supported by other specialists due to lack of evidence in Egyptology.	6, 22
The experiments have practical value that civil engineering in some areas can use kites where machinery is needed.	1, 12
Reference Summary 4	
Recently a new theory was put forth that perhaps giant kites were used in the making of the pyramids.	2, 4
The idea came from an appearance of such in a hieroglyph in an ancient book.	17, 24
This picture was given to an aeronautics professor who investigated the possibility of using kites as heavy lifters.	2, 7
The researchers found that kites could indeed lift huge weights.	3
Since there are no pictures there is no way to really show how the pyramids were really made though some clues exist that they very well could have.	11, 16, 20
Never the less, kites make sensible construction tools in the 21st century.	1

Table 5.4: Human-written reference summaries for document *Pulling string to build pyramids*, marked with the SCUs of each sentence

No.	Label	Weight
1	Kites are also useful today.	4
2	Kites could have been used.	4
3	Physically, kites are able to lift stones.	4
4	The pyramids were built.	4
5	An experiment was conducted to test the possibility of using kites.	3
6	Many specialists are still unconvinced.	3
7	The investigation was conducted by an aeronautics professor.	3
8	A 33.5-tonne column was lifted in the experiment.	2
9	A 40-square-meter nylon sail was used in the experiment.	2
10	Ancient Egyptians were interested in flying.	2
11	It remains unclear how the pyramids were built.	2
12	Kites can replace modern construction machinery.	2
13	Nobody knows how the pyramids were built.	2
14	The investigation was conducted by Morteza Gharib.	2
15	The pyramids are in Egypt.	2
16	There are no pictures showing the construction of the pyramids.	2
17	There is a hieroglyph of men using a kite.	2
18	Ancient people built the pyramids.	1
19	Kites can put up building roofs.	1
20	Some clues exist that the pyramids were built in the hypothesized way.	1
21	Some specialists are convinced.	1
22	Some specialists are doubtful due to lack of evidence in Egyptology.	1
23	The hieroglyph evidence was noticed by Maureen Clemmons.	1
24	The hypothesis came from the hieroglyph evidence.	1
25	The pyramids were built 3000 years ago.	1
26	We don't have evidence of ancient kites.	1

Table 5.5: List of SCUs derived from the reference summaries in Table 5.4

Summary Output	SCUs
PbAbs	
The pyramids of Egypt were built and no one knows.	4, 13*, 15
They were holding looked.	
The men were using it to lift a object.	
Intrigued, by Clemmons, Morteza Gharib was contacted.	14*, 23*
I have an interest in science. he says.	14*
Investigating the possibility of using kites, as lifters.	2, 5
There is the task of raising no source of energy except the wind, in a column from using.	
The kite lifted the column clean off the ground.	3
Could have used to lift stones into place.	2*
The evidence for using.	
Sailors like the Egyptians.	
They know to have used pulleys.	
As early as 1250 B.C.,	
Seq2seq	
Software consultant called maureen clemmons has suggested kites might have been involved.	2
While perusing a book on the monuments of egypt, she noticed a hieroglyph that showed a row of men standing in odd postures.	17*, 23
She wondered if perhaps the bird was actually a giant kite, and the men were using it to lift a heavy object.	17*
Earlier this year, the team put clemmons's unlikely theory to the test, using a 40-square-meter rectangular nylon sail.	5, 9
The wind was blowing at a gentle 16 to 20 kilometers an hour, little more than half what they thought would be needed.	

Table 5.6: System summaries for the example shown in Table 5.4 and 5.5, marked with the reference SCUs identified in each sentence

System		**Documents**						
		I	II	III	IV	V	VI	Average
PbAbs	Pyramid score/%	41	44	23	8	37	50	34
	Avg SCU weight	2.9	2.4	2.3	2.0	2.3	3.0	2.5
Seq2seq	Pyramid score/%	11	26	30	8	31	28	22
	Avg SCU weight	1.5	2.5	1.6	2.0	2.1	2.4	2.0

Table 5.7: Pyramid score (in percentage) and average SCU weight of PbAbs and Seq2seq on six documents

	Grammatical			Truthful		
	PbAbs	PbTok	Tie	PbAbs	PbTok	Tie
Average	20.0	5.0	15.0	25.3	5.3	9.3
Total	120	30	90	152	32	56

Table 5.8: Text quality evaluation: number of "better" judgements given

This problem is due to the fact that most modifications are represented by their own propositions, which are individually selectable. Furthermore, because memory selection takes place after processing each sentence, propositions from the same sentence are subject to direct competition. On the other hand, there is the problem of content redundancy, as PbAbs does not have a mechanism to identify and merge propositions of similar meanings across different sentences.

5.4.3.3 Human evaluation for language generation

An additional experiment concerns the text quality of the ERG-based abstractive generation, which is conducted by my collaborator Zhu. This experiment is a human evaluation of pairs of individual sentences that would be generated by PbAbs and PbTok for the same propositions. Six human subjects were asked to evaluate the same 40 pairs of sentences in randomized order, on the following two standards:

1. Which sentence is more grammatical?

2. Which sentence is more truthful to the meaning of the source sentence?

The source sentences are provided for reference. The evaluators were allowed to judge two generations as equally good (i.e. tied).

The results of this experiment are provided in Table 5.8. The PbAbs generations are judged as more grammatical or more truth-preserving in half or more than half of the pairs, respectively. The advantage of PbAbs over PbTok is statistically significant under the sign test ($p < 0.01$) in both criteria. There are only few cases where PbTok leads to better quality than PbAbs, which are mostly due to parsing mistakes made by the ACE processor.

What is left unexplored in creating our abstractive system is the possibility of synthesizing a sentence to express propositions which originate from different source sentences. Although abstractive generation should in principle be agnostic

to the surface form of the input, the current approach still makes a proposition dependent on its original sentence as "raw material". As a result, the summary sentences by PbAbs are often short and resemble bullet points. In contrast, Seq2seq generates comparatively longer sentences, but most are still subsequences extracted from the source text. On the very few occasions when it fuses information from different sentences together, it is often at a huge cost of truthfulness. For example, it generated the following (otherwise fluent) sentence, where it matched one invention with the description of another invention, and attributed it wrongly to Paleolithic times:

> *Percussion methods of fire-lighting date back to Paleolithic times, when a group of French chemists came up with the Phosphoric Candle or Ethereal Match, which contained potassium chlorate with a relatively high ignition temperature of 182 degrees centigrade.*

To correctly synthesize across sentences would require a discourse model that can plan the order and structure of the expression of summary propositions, for example to aggregate propositions about the same discourse referent, and to generate appropriate referring expressions. Similar to the creation of macropropositions, these operations are too technologically complex to be included in this project, but are nonetheless possible improvements once the foundation is laid.

Chapter 6

Conclusions

6.1 Contributions

The main contribution of my thesis is to provide the first implementation of the KvD model. During the course of my implementation, I have devised a set of procedures which operationalize the memory cycles. I have had to make both simplifications and additions to the model so that it can be run with existing technologies on actual texts. I have identified argument overlap as the bottleneck of the intelligent processing of propositions, and have experimented with different ways of modelling argument overlap.

The advantage of using a model of comprehension for summarization is its inherent explanatoriness. The resulting summaries are not based on a single-step optimization or prediction, but are the end result of a series of meaningful modules. As demonstrated by the experiment of argument overlap models, replacing these modules is relatively straightforward. This means my system can benefit from advances of upstream NLP technologies such as syntactic and semantic parsing, coreference resolution, and distributional semantics. In return, my summarizer also serves as a test bed for these NLP systems.

My summarizer significantly outperforms the existing summarizers I have tested. This is remarkable because my summarizer is not intentionally optimized for performance, but is created to test the potential of a new methodology for summarization. Using the experiments, I have validated the proposition-based incremental processing model, and also proposition-based generation.

6.2 Limitations and possible improvements

One of the limitations which I have realized during the early stage of developing my system is the lack of a perfect way to model argument overlap. As I have described in Chapter 4, existing coreference resolution systems often perform well at resolving names and pronouns, but struggle to solve anything that requires semantics, knowledge or inference. Because the gold standard mostly covers the easy cases, this problem does not catch enough attention until downstream applications (such as my summarizer) require information beyond the easy cases. In order to recover the missing links that are supposedly used by readers to establish coherence, I had to resort to lexical similarity or distributional similarity, which are also far from perfect solutions to argument overlap. Because arguments can also overlap with entire propositions, the solution of argument overlap ultimately depends on a better model of compositional semantics, potentially by combining symbolic and sub-symbolic representations.

Argument overlap is not only practically difficult, but is also theoretically difficult because it is currently studied as different tasks such as coreference resolution, paraphrase detection, and word similarity/association. There is not a unified model, because such a model would equate to a model of the complete semantic knowledge of an average person. However, it may be possible to annotate argument overlap in a document, for example in the form of a generalized version of coreference annotation.

Another possible area of improvement is the leading-edge strategy. This algorithm uses a combination of recency and generality requirements, but is overall unsophisticated. A better algorithm of selecting propositions for memory retention may necessitate changes to the memory tree. For example, intuitively, propositions which express events should have a different status from propositions which express time, location, or other general modifications. Additionally, the improved algorithm does not have to use the binary distinction between activated and deactivated nodes, but could use a continuous activation value, which corresponds to the ease of access in memory. As a positive side effect of this, memory counts will become continuous, which would mean more distinctiveness between propositions during the ranking stage.

I also speculate the possibility of fully trainable memory operations, including attachment, forgetting, and other transformations. A potential difficulty arises

from the fact that, if the memory operations remain discrete, such training has to rely on reinforcement learning, which is potentially more difficult to train than supervised learning. But this attempt would be rewarding in two aspects: First, we may test new memory operations and discover a strategy that leads to better summarization than the current one. For example, if propositions are represented as vectors, a "macro-operation" can be a neural network which computes a new proposition vector from multiple input propositions. Second, because the resulting model would still be explanatory, researchers can compare the system's behaviour (sequence of actions) against the KvD model or other theories of comprehension.

Finally, my system could be improved by additionally modelling global coherence. The processing of long, structured texts would benefit most from a model of global coherence. Macrostructure is not currently implemented because (1) the necessary generalization and inference mechanisms are beyond current NLP technologies, and (2) macrostructure is specific to a particular domain or genre of texts, which makes it less interesting.

However, it may be possible to simulate the effect of global coherence without actually implementing KvD's macrostructure. For example, all propositions of a document can be clustered into several locally coherent groups. Then, a specifically-made RST parser can be used to identify the rhetorical relations between these clusters and create a discourse tree. The question remains to be answered is the interaction between the RST tree and the memory tree. (If they are run independent of each other, then this is essentially running two summarizers and combining their results.)

In this thesis, I have shown that even just one of KvD's core ideas (incremental processing of propositions to simulate local coherence) already results in effective and explanatory summarization. If we were able to also implement the second core idea of the model, i.e. global coherence, then even stronger summarization capacity should result.

Bibliography

David J Allerton. The notion of givenness and its relations to presupposition and to theme. *Lingua Amsterdam*, 44(2-3):133–168, 1978.

Chinatsu Aone, Mary Ellen Okurowski, James Gorlinsky, and Bjornar Larsen. A scalable summarization system using robust NLP. *Intelligent Scalable Text Summarization*, 1997.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.

Ayana, Shiqi Shen, Zhiyuan Liu, and Maosong Sun. Neural headline generation with minimum risk training. *arXiv preprint arXiv:1604.01904*, 2016.

Alan Baddeley. *Working memory, thought, and action.* Oxford University Press, 2007.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, 2015.

Simon Baker, Roi Reichart, and Anna Korhonen. An unsupervised model for instance level subcategorization acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 278–289, Doha, Qatar, October 2014.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014.

Regina Barzilay and Michael Elhadad. Using lexical chains for text summarization. *Advances in automatic text summarization*, pages 111–121, 1999.

Regina Barzilay and Mirella Lapata. Modeling local coherence: An entity-based approach. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 141–148, Ann Arbor, Michigan, June 2005.

Sandro Bauer and Simone Teufel. A methodology for evaluating timeline generation algorithms based on deep semantic units. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 834–839, Beijing, China, July 2015.

Yves Bestgen, Guy Lories, and Jennifer Thewissen. Using latent semantic analysis to measure coherence in essays by foreign language learners? In *Proceedings of 10th International Conference on statistical analysis of textual data (JADT)*, pages 385–395, 2010.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

Anders Björkelund and Richárd Farkas. Data-driven multilingual coreference resolution using resolver stacking. In *Joint Conference on EMNLP and CoNLL – Shared Task*, pages 49–55, Jeju Island, Korea, July 2012.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

Ted Briscoe, John Carroll, and Rebecca Watson. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 77–80, Sydney, Australia, July 2006.

Claire Cardie and Kiri Wagstaf. Noun phrase coreference as clustering. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June 2005.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014.

Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany, August 2016.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

Kevin Clark and Christopher D. Manning. Improving coreference resolution by learning entity-level distributed representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 643–653, Berlin, Germany, August 2016.

James Clarke and Mirella Lapata. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429, 2008.

Trevor Cohn and Mirella Lapata. Large margin synchronous generation and its application to sentence compression. In *Proceedings of the 2007 Joint Conference*

on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 73–82, Prague, Czech Republic, June 2007.

Trevor Cohn and Mirella Lapata. Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 137–144, Manchester, UK, August 2008.

John M Conroy and Dianne P O'leary. Text summarization via hidden Markov models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 406–407. ACM, 2001.

Ann Copestake. Slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 1–9, Athens, Greece, March 2009.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2-3): 281–332, 2005.

Alfred Correira. Computing story trees. *Computational Linguistics*, 6(3-4):135–149, 1980.

Aron Culotta, Michael Wick, and Andrew McCallum. First-order probabilistic models for coreference resolution. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 81–88, Rochester, New York, April 2007.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa, 2006.

Gerald DeJong. An overview of the FRUMP system. *Strategies for natural language processing*, 113:149–176, 1982.

Katherine A DeLong, Thomas P Urbach, and Marta Kutas. Probabilistic word pre-activation during language comprehension inferred from electrical brain activity. *Nature neuroscience*, 8(8):1117, 2005.

Pascal Denis and Jason Baldridge. Joint determination of anaphoricity and coreference resolution using integer programming. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 236–243, Rochester, New York, April 2007.

Pascal Denis and Jason Baldridge. Specialized models and ranking for coreference resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 660–669, Honolulu, Hawaii, October 2008.

Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics*, 19(1):61–74, 1993.

Greg Durrett and Dan Klein. Easy victories and uphill battles in coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1971–1982, Seattle, Washington, USA, October 2013.

Gonenc Ercan and Ilyas Cicekli. Lexical cohesion based topic modeling for summarization. In *Computational Linguistics and Intelligent Text Processing*, pages 582–592. Springer, 2008.

Günes Erkan and Dragomir R Radev. LexRank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, pages 457–479, 2004.

Tobias Falke, Christian M. Meyer, and Iryna Gurevych. Concept-map-based multi-document summarization using concept coreference resolution and global importance optimization. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 801–811, Taipei, Taiwan, November 2017.

Yimai Fang and Simone Teufel. A summariser based on human memory limitations and lexical competition. In *Proceedings of the 14th Conference of*

the European Chapter of the Association for Computational Linguistics, pages 732–741, Gothenburg, Sweden, April 2014.

Yimai Fang and Simone Teufel. Improving argument overlap for proposition-based summarisation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Berlin, Germany, August 2016.

Yimai Fang, Haoyue Zhu, Ewa Muszyńska, Alexander Kuhnle, and Simone Teufel. A proposition-based abstractive summariser. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 567–578, Osaka, Japan, December 2016.

Elena Filatova and Vasileios Hatzivassiloglou. A formal model for information selection in multi-sentence text extraction. In *Proceedings of COLING 2004*, pages 397–403, Geneva, Switzerland, Aug 23–Aug 27 2004.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM, 2001.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. Generation from abstract meaning representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 731–739, San Diego, California, June 2016.

Dan Flickinger. On building a more effcient grammar by exploiting types. *Natural Language Engineering*, 6(01):15–28, 2000.

Dan Flickinger, Emily M. Bender, and Stephan Oepen. Towards an encyclopedia of compositional semantics. Documenting the interface of the English Resource Grammar. In *Proceedings of the Ninth Language Resources and Evaluation Conference (LREC '14)*, pages 875–881, Reykjavik, Iceland, 2014.

Peter W Foltz, Walter Kintsch, and Thomas K Landauer. An analysis of textual coherence using latent semantic indexing. In *Third Annual Conference of the Society for Text and Discourse*, Boulder, Colorado, 1993.

Peter W Foltz, Walter Kintsch, and Thomas K Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3): 285–307, 1998.

Gottlob Frege. Sense and reference. *The philosophical review*, 57(3):209–230, 1948.

Maria Fuentes, Enrique Alfonseca, and Horacio Rodríguez. Support vector machines for query-focused summarization trained and evaluated on pyramid data. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 57–60, Prague, Czech Republic, June 2007.

Danilo Fum, Giovanni Guida, and Carlo Tasso. Forward and backward reasoning in automatic abstracting. In *Proceedings of the 9th Conference on Computational linguistics (Volume 1)*, pages 83–88. Academia Praha, 1982.

Dimitrios Galanis, Gerasimos Lampouras, and Ion Androutsopoulos. Extractive multi-document summarization with integer linear programming and support vector regression. In *Proceedings of COLING 2012*, pages 911–926, Mumbai, India, December 2012.

Michel Galley and Kathleen McKeown. Improving word sense disambiguation in lexical chaining. In *IJCAI*, volume 3, pages 1486–1488, 2003.

Pierre-Etienne Genest and Guy Lapalme. Framework for abstractive summarization using text-to-text generation. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 64–73, Portland, Oregon, June 2011.

Pierre-Etienne Genest and Guy Lapalme. ABSUM: a knowledge-based abstractive summarizer. *Génération de résumés par abstraction*, 25, 2013.

H. Paul Grice. Logic and conversation. In P. Cole and J. Morgan, editors, *Studies in Syntax and Semantics III: Speech Acts*, pages 183–198. Academic Press, 1975.

Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Providing a unified account of definite noun phrases in discourse. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 44–50, Cambridge, Massachusetts, USA, June 1983.

Barbara J Grosz, Scott Weinstein, and Aravind K Joshi. Centering: A framework for modeling the local coherence of discourse. *Computational linguistics*, 21(2): 203–225, 1995.

Jeanette K Gundel, Nancy Hedberg, and Ron Zacharski. Cognitive status and the form of referring expressions in discourse. *Language*, pages 274–307, 1993.

Surabhi Gupta, Ani Nenkova, and Dan Jurafsky. Measuring importance and query relevance in topic-focused multi-document summarization. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 193–196, Prague, Czech Republic, June 2007.

Aria Haghighi and Dan Klein. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1152–1161, Singapore, August 2009.

Aria Haghighi and Lucy Vanderwende. Exploring content models for multi-document summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 362–370, Boulder, Colorado, June 2009.

Udo Hahn and Ulrich Reimer. Computing text constituency: An algorithmic approach to the generation of text graphs. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 343–368. British Computer Society, 1984.

Michael AK Halliday and Ruqaiya Hasan. *Cohesion in English*. Longman Group Ltd, 1976.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

Frances K Heussenstamm. Bumper stickers and the cops. *Society*, 8(4):32–33, 1971.

Felix Hill, Kyunghyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. Embedding word similarity with neural machine translation. In *Workshop Paper at ICLR*, 2015.

Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 2016.

Lynette Hirschman. MUC-7 coreference task definition. Technical report, 1997. URL `http://www-nlpir.nist.gov/related_projects/muc/proceedings/co_task.html`.

Deirdre Hogan. Coordinate noun phrase disambiguation in a generative parsing model. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 680–687, Prague, Czech Republic, June 2007.

Eduard Hovy and Chin-Yew Lin. Automated text summarization and the SUMMARIST system. In *Proceedings of the TIPSTER Text Program: Phase III*, pages 197–214, Baltimore, Maryland, USA, October 1998.

Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th International Conference on Research in Computational Linguistics*, 1997.

Daniel Jurafsky and James H Martin. *Speech and Language Processing*. Prentice-Hall, Inc., second edition, 2009.

J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. Derivation of new readability formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for navy enlisted personnel. 1975.

Alison King. Comparison of self-questioning, summarizing, and notetaking-review as strategies for learning from lectures. *American Educational Research Journal*, 29(2):303–323, 1992.

Walter Kintsch. *The representation of meaning in memory*. Lawrence Erlbaum, 1974.

Walter Kintsch. The role of knowledge in discourse comprehension: A construction-integration model. *Psychological review*, 95(2):163, 1988.

Walter Kintsch and Teun A van Dijk. Toward a model of text comprehension and production. *Psychological review*, 85(5):363, 1978.

Walter Kintsch and Douglas Vipond. Reading comprehension and readability in educational practice and psychological theory, 1979.

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July 2003.

Kevin Knight and Daniel Marcu. Statistics-based summarization – step one: Sentence compression. *AAAI/IAAI*, 2000:703–710, 2000.

Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1): 91–107, 2002.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (Volume 1)*, pages 48–54, 2003.

Julian Kupiec, Jan Pedersen, and Francine Chen. A trainable document summarizer. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 68–73. ACM, 1995.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

Sebastian Padóand Mirella Lapata. Constructing semantic space models from parsed corpora. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Sapporo, Japan, July 2003.

Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, pages 24–26. ACM, 1986.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational*

Linguistics (Volume 2: Short Papers), pages 302–308, Baltimore, Maryland, June 2014.

Wei Li, Lei He, and Hai Zhuge. Abstractive news summarization based on event semantic link network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 236–246, Osaka, Japan, December 2016.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July 2004.

Chin-Yew Lin and Eduard Hovy. The automated acquisition of topic signatures for text summarization. In *Proceedings of the 18th conference on Computational linguistics (Volume 1)*, pages 495–501, 2000.

Chin-Yew Lin and Eduard Hovy. The potential and limitations of automatic sentence extraction for summarization. In Dragomir Radev and Simone Teufel, editors, *Proceedings of the HLT-NAACL 03 on Text summarization workshop-Volume 5*, pages 73–80, 2003.

Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 768–774, Montreal, Quebec, Canada, August 1998.

Yang Liu and Mirella Lapata. Learning structured text representations. *Transactions of the Association of Computational Linguistics*, 6:63–75, 2018.

Hector Llorens, Estela Saquete, and Borja Navarro. TIPSem (English and Spanish): Evaluating CRFs and semantic roles in TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, Uppsala, Sweden, July 2010.

Annie Louis and Ani Nenkova. Automatically evaluating content selection in summarization without human models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 306–314, Singapore, August 2009.

Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

Xiaoqiang Luo, Abe Ittycheriah, Hongyan Jing, Nanda Kambhatla, and Salim Roukos. A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 135–142, Barcelona, Spain, July 2004.

William C Mann and Sandra A Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281, 1988.

Daniel Marcu. To build text summaries of high quality, nuclearity is not sufficient. In *Working Notes of the AAAI-98 Spring Symposium on Intelligent Text Summarization*, pages 1–8, 1998.

Marie-Catherine de Marneffe and Christopher D Manning. Stanford typed dependencies manual. Technical report, 2008. URL `http://nlp.stanford.edu/software/dependencies_manual.pdf`. Revised in 2013.

André FT Martins and Noah A Smith. Summarization with a joint model for sentence extraction and compression. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, pages 1–9, 2009.

Sebastian Martschat and Michael Strube. Recall error analysis for coreference resolution. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2070–2081, Doha, Qatar, October 2014.

Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In *Advances in neural information processing systems*, pages 905–912, 2005.

Ryan McDonald. A study of global inference algorithms in multi-document summarization. *Advances in Information Retrieval*, pages 557–564, 2007.

Ryan T McDonald. Discriminative sentence compression with soft syntactic evidence. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

Rada Mihalcea and Paul Tarau. TextRank: Bringing order into texts. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 404–411, Barcelona, Spain, July 2004.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.

Jane Morris and Graeme Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational linguistics*, 17(1):21–48, 1991.

Preslav Nakov, Antonia Popova, and Plamen Mateev. Weight functions impact on LSA performance. *EuroConference RANLP*, pages 187–193, 2001.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, 2016.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. SummaRuNNer: A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pages 3075–3081, 2017.

Neha Nayak, Mark Kowarsky, Gabor Angeli, and Christopher D Manning. A dictionary of nonsubsective adjectives. Technical report, Technical Report CSTR 2014-04, Department of Computer Science, Stanford University, October 2014.

Ani Nenkova and Kathleen McKeown. Automatic summarization. *Foundations and Trends® in Information Retrieval*, 5(2–3):103–233, 2011.

Ani Nenkova and Rebecca Passonneau. Evaluating content selection in summarization: The pyramid method. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 145–152, Boston, Massachusetts, USA, May 2 - May 7 2004.

Ani Nenkova and Lucy Vanderwende. The impact of frequency on summarization. Technical report, Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-2005-101, 2005.

Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.

Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Philadelphia, Pennsylvania, USA, July 2002.

Hitoshi Nishikawa, Kazuho Arita, Katsumi Tanaka, Tsutomu Hirao, Toshiro Makino, and Yoshihiro Matsuo. Learning to generate coherent summary with discriminative hidden semi-Markov model. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1648–1659, Dublin, Ireland, August 2014.

Joakim Nivre, Johan Hall, and Jens Nilsson. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219, 2006.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666, 2016.

Franz Josef Och, Christoph Tillmann, Hermann Ney, et al. Improved alignment models for statistical machine translation. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28, 1999.

Ruth O'Donovan, Michael Burke, Aoife Cahill, Josef Van Genabith, and Andy Way. Large-scale induction and evaluation of lexical resources from the Penn-II and Penn-III Treebanks. *Computational Linguistics*, 31(3):329–366, 2005.

Charles Kay Ogden and Ivor Armstrong Richards. The meaning of meaning: A study of the influence of thought and of the science of symbolism. 1923.

Kenji Ono, Kazuo Sumita, and Seiji Miike. Abstract generation based on rhetorical structure extraction. In *Proceedings of the 15th conference on Computational linguistics (Volume 1)*, pages 344–348, 1994.

Miles Osborne. Using maximum entropy for sentence extraction. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, pages 1–8, Phildadelphia, Pennsylvania, USA, July 2002.

Allan Paivio. *Mental representations: A dual coding approach.* Oxford University Press, 1990.

Aannemarie Sullivan Palinscar and Ann L Brown. Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and instruction*, 1(2):117–175, 1984.

Rebecca J. Passonneau, Emily Chen, Weiwei Guo, and Dolores Perin. Automated pyramid scoring of summaries using distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 143–147, Sofia, Bulgaria, August 2013.

Haoruo Peng, Daniel Khashabi, and Dan Roth. Solving hard coreference problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 809–819, Denver, Colorado, May–June 2015.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on*

Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar, October 2014.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006.

Judita Preiss, Ted Briscoe, and Anna Korhonen. A system for large-scale acquisition of verbal, nominal and adjectival subcategorization frames from corpora. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 912–919, Prague, Czech Republic, June 2007.

Ellen F Prince. Toward a taxonomy of given-new information. In *Radical pragmatics*. Academic Press, 1981.

Ellen F Prince. The ZPG letter: Subjects, definiteness, and information-status. *Discourse description: diverse analyses of a fund raising text*, pages 295–325, 1992.

Dragomir Radev, Timothy Allison, Sasha Blair-Goldensohn, John Blitzer, Arda Celebi, Stanko Dimitrov, Elliott Drabek, Ali Hakim, Wai Lam, Danyu Liu, et al. MEAD-a platform for multidocument multilingual text summarization. In *Proceedings of the International Conference on Language Resources and Evaluation*, 2004.

Dragomir R Radev and Kathleen R McKeown. Generating natural language summaries from multiple on-line sources. *Computational Linguistics*, 24(3): 470–500, 1998.

Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nate Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501, Cambridge, MA, October 2010.

Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

Cody Rioux, Sadid A. Hasan, and Yllias Chali. Fear the REAPER: A system for automatic multi-document summarization with reinforcement learning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 681–690, Doha, Qatar, October 2014.

Rachel Rudinger and Benjamin Van Durme. Is the Stanford dependency representation semantic? In *Proceedings of the Second Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 54–58, 2014.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, 2015.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017.

Dou Shen, Jian-Tao Sun, Hua Li, Qiang Yang, and Zheng Chen. Document summarization using conditional random fields. In *IJCAI*, volume 7, pages 2862–2867, 2007.

H. Gregory Silber and Kathleen F. McCoy. Efficiently computed lexical chains as an intermediate representation for automatic text summarization. *Computational Linguistics*, 28(4):487–496, December 2002.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August 2013.

Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, 27(4):521–544, 2001.

Karen Spärck Jones. Automatic summarizing: factors and directions. *Advances in automatic text summarization*, pages 1–12, 1999.

Gabriel Stanovsky and Ido Dagan. Creating a large benchmark for Open Information Extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2300–2305, Austin, Texas, November 2016.

Michael Strube and Udo Hahn. Functional centering: Grounding referential coherence in information structure. *Computational linguistics*, 25(3):309–344, 1999.

Simone Teufel and Marc Moens. Sentence extraction as a classification task. In *ACL/EACL-97 Workshop on Intelligent Scalable Text Summarization*, pages 58–65, Madrid, Spain, July 1997.

Simone Teufel and Marc Moens. Summarizing scientific articles: experiments with relevance and rhetorical status. *Computational linguistics*, 28(4):409–445, 2002.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–85, Berlin, Germany, August 2016.

Althea Turner and Edith Greene. The construction and use of a propositional text base. Technical report, Department of Psychology, University of Colorado, 1977.

Peter D Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. *European Conference on Machine Learning*, pages 491–502, 2001.

Kees Van Deemter and Rodger Kibble. On coreferring: Coreference in muc and related annotation schemes. *Computational linguistics*, 26(4):629–637, 2000.

Hans van Halteren and Simone Teufel. Examining the consensus between human summaries: initial experiments with factoid analysis. In Dragomir Radev and Simone Teufel, editors, *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 57–64, 2003.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

Xun Wang, Yasuhisa Yoshida, Tsutomu Hirao, Katsuhito Sudoh, and Masaaki Nagata. Summarization based on task-oriented discourse parsing. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(8):1358–1367, 2015.

Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, et al. Co-reference Guidelines for English OntoNotes. Technical report, 2011. URL `https://catalog.ldc.upenn.edu/docs/LDC2011T03/coreference/english-coref.pdf`.

Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1): 1–191, 1972.

Kam-Fai Wong, Mingli Wu, and Wenjie Li. Extractive summarization using supervised and semi-supervised learning. In *Proceedings of the 22nd International Conference on Computational Linguistics (Volume 1)*, pages 985–992, 2008.

Jin-ge Yao, Xiaojun Wan, and Jianguo Xiao. Recent advances in document summarization. *Knowledge and Information Systems*, 53(2):297–336, 2017.

Katsumasa Yoshikawa, Ryu Iida, Tsutomu Hirao, and Manabu Okumura. Sentence compression with semantic role constraints. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 349–353, Jeju Island, Korea, July 2012.

Renxian Zhang, Wenjie Li, Naishi Liu, and Dehong Gao. Coherent narrative summarization with a cognitive model. *Computer Speech & Language*, 35: 134–160, 2016.

Zhi Zhong and Hwee Tou Ng. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of the ACL 2010 System Demonstrations*, pages 78–83, Uppsala, Sweden, July 2010.

Rolf A Zwaan. The immersed experiencer: Toward an embodied theory of language comprehension. *Psychology of learning and motivation*, 44:35–62, 2003.

Appendix A

Proposition building

This appendix describes my method of building KvD-style propositions from grammatical dependencies. The propositions used by KvD can be categorized into three types (Turner and Greene, 1977):

Predicate propositions A predicate proposition is one that expresses an action or a relation. In addition to typical events such as subject–verb–object triplets, this type also includes nominal propositions, which express set membership, and referential propositions, which express identity of referents.

Modifier propositions Modifier propositions mainly include the use of qualifiers (typically adjectives and adverbs), quantifiers, partitives, and negations.

Connective propositions Connective propositions include conjunctions, disjunctions, and propositions that express causality, purpose, concession, contrast, condition, and circumstance (including time, location, and manner).

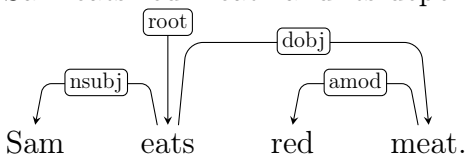
The source from which propositions are created is the Stanford Dependencies (SD; de Marneffe et al., 2006), which is a dependency grammar widely used in NLP tasks such as information extraction and event extraction. Other dependency grammars, including its cross-lingual successor called the Universal Dependencies (Nivre et al., 2016), define similar types of dependencies. SD can be obtained using phrase-structure parsers, such as Klein and Manning (2003), Petrov et al. (2006), and Socher et al. (2013), and dependency parsers, such as Nivre et al. (2006), and Chen and Manning (2014). Because the unit size of a dependency

is as small as a pair of words, the creation of a proposition typically requires aggregating multiple dependencies.

SD uses approximately 50 relation types, which are classified into a specificity hierarchy. The most generic type, dependent (abbreviated `dep`), is divided into 10 subtypes: auxiliary (`aux`), argument (`arg`), coordination (`cc`), conjunct (`conj`), expletive (`expl`), modifier (`mod`), parataxis (`parataxis`), punctuation (`punct`), referent (`ref`), and semantic dependent (`sdep`).[1] Some of these subtypes are further subdivided. A parser should use the most specific type in the hierarchy that is appropriate, and fall back to the most generic type (`dep`) when type cannot be determined (which could be caused by unusual grammatical constructions or parsing errors).

Proposition building can be described as a series of manipulations operating on the dependency trees of the input sentences. Starting at the root node of a sentence (indicated by the `root` relation), I create propositions from that node and repeat the process on each of its children. This corresponds to a pre-order traversal of the dependency tree.

I will now illustrate the basic process to create propositions using an example. Consider the sentence "*Sam eats red meat*" and its dependency parse:



The root of the dependency tree is "*eats*". For this node, we gather all its dependents that stand in an argument relation with it. It has the subject "*Sam*" and the direct object "*meat*". Thus a predicate proposition `eat (Sam, meat)` is created, and we move to its two child nodes. Because "*Sam*" does not have any dependents, no new propositions are created from it. "*Meat*" has an adjectival modifier "*red*", hence a modifier proposition `red (meat)` is created, using the modifier as the predicate. This concludes the tree traversal and leaves us with the two propositions:

```
eat (Sam, meat)
red (meat)
```
(A.1)

[1] The relation types, which are defined by Marneffe and Manning (2008), do not necessarily coincide with a linguistic classification of grammatical relations. Therefore, the names of the types should be regarded as technical terms of SD.

This basic process of tree traversal is preceded by the preprocessing of the dependency tree, and is followed by the post-processing of the propositions created, as illustrated in Figure A.1. During preprocessing, I merge nodes that are connected by edges of type multi-word expression, determiner, quantifier, negation, and auxiliary (step a). I also create imaginary nodes to represent coordination (step b). In the post-processing of propositions, I distribute any proposition over its coordinated predicate or argument, i.e. distributing $P(A_1 \wedge A_2, B)$ over $A_1 \wedge A_2$ produces $P(A_1, B) \wedge P(A_2, B)$ (steps c, d). I create embedded propositions by replacing each argument of a proposition which is a predicate of another proposition with that proposition (textually represented by the proposition number following a # symbol) (step e). Finally, I reorder the propositions by the textual location of their predicates (step f). The order is important to KvD, as they process propositions in this order. But in my model, the processing order of propositions of the same sentence is determined by the strength of argument overlap; the proposition order is merely a tie-breaker.

So far I have assumed using the basic representation of SD, where the dependency graph is a tree, and every token of a sentence is a node. In fact, the Stanford parser (Klein and Manning, 2003), which I use in the experiments, is also capable of outputting two variant representations that are rule-based transformations of the basic dependencies. Motivated by information extraction applications, many of these transformations create (sometimes non-tree or non-projective) edges between content words in order to improve the recall of semantic relations. However, the transformations do not fully satisfy the requirement of creating propositions, and have many side-effects such as compromises for tokens that are no longer nodes. What is achieved by these transformations is often based on circumstances. For instance, the propagation of subject for a conjunction predicate is a low-hanging fruit, but it is not fundamentally different from the propagation of object. It is not linguistically justified to favour one over the other. The method I present in this appendix aims to provide a systematic framework of converting basic dependencies to propositions, which could be extended and improved in any area as necessary, by using linguistic knowledge or data-driven methods.

In the following sections, I will discuss aspects of the proposition building process that are related to some well-known semantic problems. Section A.1 focuses on step (a). Sections A.2 and A.4 contain considerations and technical details that mainly surround the tree traversal stage. Section A.3 is useful to

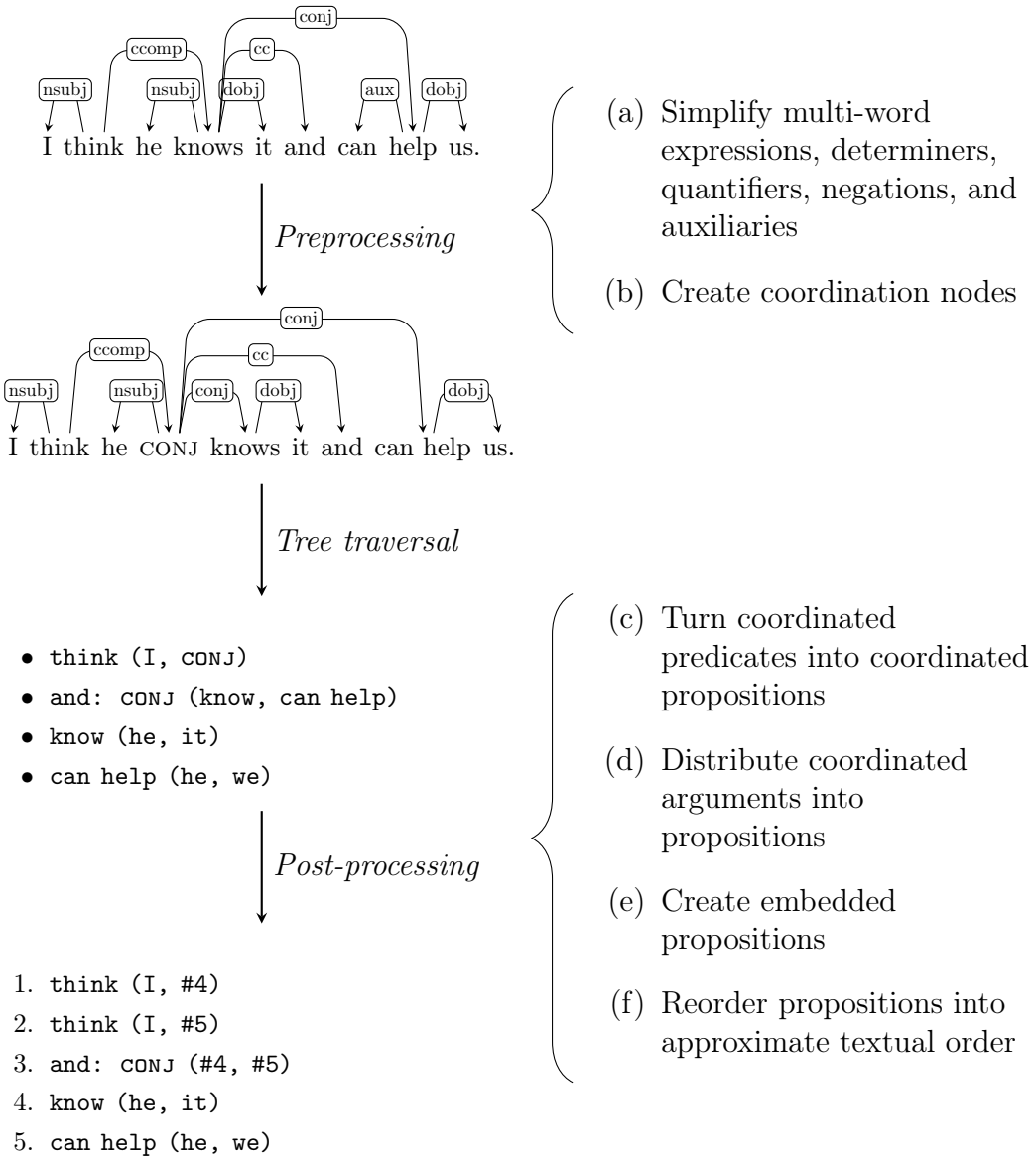


Figure A.1: Steps of creating propositions from dependencies

understand steps (b, c, d). Finally, Section A.5 summarizes the outcome of all dependency types.

A.1 Merging tokens

The most prominent difference between my propositions and KvD's propositions is that my propositions use textual tokens to stand for concepts. In the KvD model, the assimilation of textual tokens into concepts is performed online during the reading process. This includes the resolution of referential equality, in which implicit referential propositions are used as intermediates. KvD do not provide an algorithmic description of the assimilation process, but only present the resulting propositions, which contain already-resolved concepts, as a device of illustration. It is the luxury of manually created concepts that allows them to determine concept overlap simply by noticing whether two concepts are written identically in capital letters.

In my model, a proposition is still an independent unit of information, but appears physically as a bag of tokens (in this thesis, represented by the lemmas of the words). The latent concept of a token is indirectly specified by the token's relation with other tokens, as opposed to explicitly labelled using human intelligence. For example, a token-based proposition `trust (Mary, he)` could stand for the KvD proposition TRUST (MARY, JOHN). This can be the case if, for example, the referent of the token `he` ("*him*" in the text) is a person who a human reader would label as JOHN. Like variable names, the label is immaterial to the summarization algorithm, as long as the relation between tokens can be established (in this example, the coreference relation between the pronoun "*him*" and the antecedent "*John*").

While in many cases one token corresponds to one concept, there are three scenarios in which multiple tokens have to be treated together as one concept. During preprocessing, the multiple nodes in the dependency tree that correspond to these tokens are replaced by a merged node. I will now discuss the three scenarios in their respective subsections.

A.1.1 Multi-word units

First, if a lexical item is expressed using several tokens, these parts shall be merged into one node in the dependency tree. This corresponds to collapsing dependency edges of type multi-word expression (`mwe`), noun compound (`nn`), phrasal verb particle (`prt`), or part of compound number (`number`) in SD. In this thesis, I denote a merged token by concatenating its components using underscores, e.g. `traffic_light`, and `take_off`.

There are also some multi-word prepositions and coordinators (e.g. "*as well as*") that the Stanford parser converts into equivalent dependency labels (e.g. `conj_and`) in its collapsed representation. I detect them according to the list by Marneffe and Manning (2008), and merge the corresponding nodes even if the types of dependencies connecting them are not one of the types listed above.

A.1.2 Function words

Second, function words which I consider not qualified for independent concepts are folded into their governors. But unlike components of a multi-word unit, which are in equal relations to each other, the function words are regarded as properties affiliated to the content words. This has significance in the computation of argument overlap, which includes an entire multi-word unit for determining the semantic similarity between two arguments, but ignores the affiliated function words. Notationally, however, affiliated words are printed together with content words in their textual order (separated by spaces), e.g. `can help`, except that articles and auxiliary `be`, `to`, `do`, and `have` are omitted. The following dependency types are affected by this process:

Determiner (`det`) and predeterminer (`predet`): I found that determiners in SD include articles such as "*the*" and "*a*", quantifiers such as "*every*", "*any*", "*all*", and other determiners such as "*another*", "*which*". A predeterminer is a quantifier that occurs before a determiner, such as "*all*" in "*all the boys*". Joining a determiner or predeterminer with the content word mimics KvD's treatment of quantifiers, in which a quantified concept is always referred to with quantification.

Numeric quantifier (`num`) and numeric quantifier modifier (`quantmod`): Similar to `det` and `predet`, this pair is also a form of quantification. For example,

in "*about 200 people*", "*200*" is a `num`, and "*about*" is a `quantmod`.

Negation (`neg`): I observed that in SD, `neg` relations can occur with verbs ("*not*" and "*n't*") and with nouns ("*no*"). The range of negative words is narrowly defined, and negations expressed by other adverbs (such as "*hardly*") are not captured by `neg`. My treatment of negation differs from that of KvD, who would create two propositions for the statement "*I didn't have breakfast*":

$$\begin{aligned} \alpha &= \text{HAVE (I, BREAKFAST)} \\ \beta &= \text{NOT } (\alpha) \end{aligned} \tag{A.2}$$

Because the risk of making it possible for the later stage to select an incorrect proposition outweighs the benefit of flexibility, I create only one proposition by adding negation to the main predicate, i.e.

```
not have (I, breakfast)
```
(A.3)

Auxiliary (`aux`) and passive auxiliary (`auxpass`): I found that auxiliaries in SD include model verbs such as "*can*" and "*should*", as well as auxiliary verbs such as "*be*" (used to form the continuous tense or the passive voice) and "*have*" (used to form the perfect tense). Such auxiliaries do not exist at all in KvD's propositions, which abstract away from tense, voice, or aspect. Kintsch treats linguistic variations such as tense and definiteness as selected by the speaker depending on context and pragmatics. This model could in theory be simulated by keeping track of the time of events, the discourse focus, and the information-status of entities, and choosing the textual realization accordingly. Because auxiliaries and articles are lexical means to express these properties (in addition to the non-lexical means of inflection), they do not qualify as independent concepts and hence are stored as affiliated tokens.

A.1.3 Non-subsective adjectives

There is a third phenomenon that challenges the token–concept correspondence, namely the mode of composition of a modifier. Most notably, there is the distinction between subsective and non-subsective adjectives. The definition of

subsective adjectives is as follows: The set of entities denoted by the combination of a subsective adjective and a noun is a subset of the set of entities denoted by the noun. For instance, a "*skilful surgeon*" is a "*surgeon*", and "*skilful*" is a property of a "*skilful surgeon*". In contrast, a non-subsective adjective makes statement about the relationship between the property expressed by the noun and the referent. For instance, "*Peter is an alleged murderer*" does not entail "*Peter is a murderer*". It is the assignment of the label "*murderer*" to Peter that is modified by "*alleged*"; "*alleged*" is not a property at all. In order to generate truthful summaries, we want to ensure that "*alleged murderer*" is treated as an atomic unit.

In SD, any adjectival modifier is an `amod`, regardless of whether it is subsective or non-subsective. The default action I take for an `amod` is to make a modifier proposition, which is equivalent to treating adjectives as subsective. Special treatment is required as far as non-subsective adjectives are concerned. I detect non-subsective adjectives using the lexicon compiled by Nayak et al. (2014). It contains 60 non-subsective adjectives collected both from previous works and by using a classifier. If an adjectival modifier matches an entry in the list, it is affiliated to the noun it modifies, forming one node that corresponds to the compositional concept.

Among subsective adjectives, there is also a distinction between intersective adjectives and non-intersective adjectives. The definition of intersective adjectives is that the combination of an intersective adjective with a noun denotes the intersection of the entity set of the adjective and the entity set of the noun. For instance, "*Canadian*" is an intersective adjective, because the set of entities denoted by "*Canadian surgeon*" is the intersection of the set denoted by "*Canadian*" and the set denoted by "*surgeon*". From the statement "*Peter is a Canadian surgeon*", we could infer the proposition CANADIAN (PETER). In contrast, "*skilful*" is not intersective, therefore "*Peter is a skilful surgeon*" and "*Peter is a violinist*" together do not entail "*Peter is a skilful violinist*"; i.e. SKILFUL (PETER) is not an independent proposition. This distinction makes a difference only if inferred propositions as such were to be created, and therefore it is not treated here.

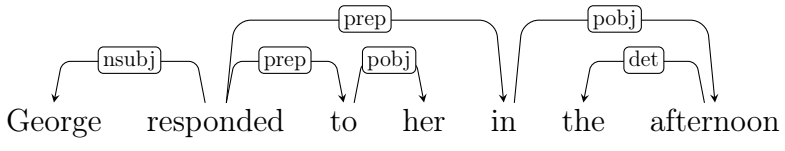
A.2 Distinguishing argument and adjunct

In linguistics, an argument completes the meaning of its predicate, whereas an adjunct provides optional information that can be removed without affecting the rest of the sentence. A general principle of proposition building is that arguments of the same predicate are packed in the same proposition, but each adjunct amounts to an extra proposition.

Most syntactic dependency grammars such as SD do not fully recover the predicate–argument structure (with the notable exception of the RASP system (Briscoe et al., 2006), which uses subcategorization attributes). In SD, only a subset of arguments are correctly labelled as argument (`arg`); this includes arguments expressed in certain constituents such as noun phrases, infinitives, gerund phrases, "*that*"- and bare clauses. In contrast, prepositional arguments are regarded as so-called modifiers in SD, making them indistinguishable from prepositional adjuncts (Rudinger and Van Durme, 2014). It is understandable because of the difficulty of distinguishing prepositional arguments from prepositional adjuncts, however it poses a problem for proposition building. Note that this problem is different from the well-known prepositional phrase (PP) attachment problem, which refers to the type of error a syntactic parser can make by attaching a PP to a wrong head (e.g. for a PP following a verb and its object, attaching to the object instead of the verb). The problem discussed here assumes that a PP is correctly attached, but it is the argument/adjunct status of the PP that is in question.

Linguistically, the number and types of arguments of a predicate is controlled by subcategorization. For instance, in "*I am proud of my daughter*", "*of my daughter*" is an argument in the subcategorization frame (SCF) for the predicate "*pround*". The identification of the SCF of a word occurrence is a research question (Baker et al., 2014). A simple baseline could be to acquire the most frequent SCFs of a lexical item from corpus data (O'Donovan et al., 2005; Preiss et al., 2007), and select the SCF of a word occurrence based on its dependencies.

However, the line between an argument and an adjunct is blurry, especially when considering optional arguments. For example, consider the following sentence:



(A.4)

In this sentence, the first PP "*to her*" is an optional argument, because "*George responded*" is still a meaningful event (hence could be a proposition); the second PP "*in the afternoon*" is an adjunct. My strategy to classify every prepositional dependency (`prep`) as either a true modifier or an argument is as follows: First, whether the governor is a predicate is determined by constituency types; verb phrase and adjectival phrase are considered to be predicates and hence can take PPs as arguments. Then, if a predicate would otherwise have no arguments (except for possibly the subject), it should take its nearest prepositional dependency as its argument. In this example, the optional argument "*to her*" would be classified as an argument of "*responded*". The motivation for this rule is the idea that propositions should contain comparable amount of information – having too many or too few arguments is discouraged. Of course, this heuristic has limitations such as being unable to handle noun predicates ("*the relation between ...*"), and mistaking PP adjuncts as arguments of intransitive verb predicates ("*sneezed in the kitchen*").

In a proposition, I store additional tokens with respect to an argument or the predicate as labels, which are textually represented as a token and a colon preceding the relevant argument or predicate. In the example given by (A.4), because the PP "*to her*" is classified as an argument, the preposition "*to*" has to be deposited as a label of "*her*". In contrast, the preposition "*in*" is a predicate because the PP "*in the afternoon*" is a modifier. The propositions resulted from the sentence are as follows:

```
#1 = respond (George, to:  her)
#2 = in (#1, afternoon)
```

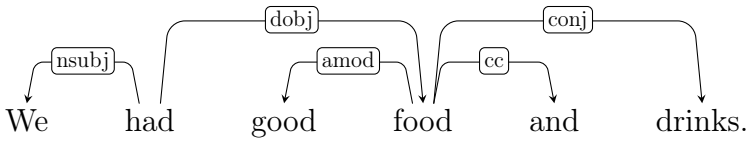
(A.5)

A.3 Handling coordination

SD represents coordination as `conj` edges from the first conjunct to every other conjunct. The first conjunct is designated the head of the coordination, which all other conjuncts are subordinate to. The coordinator, such as "*and*" or "*as well*

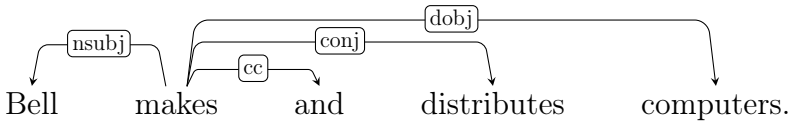
as", is attached to the coordination head using an edge of relation type `cc`. The unequal relation between the first conjunct and the other conjuncts enables SD to underspecify whether a word is a dependent of the first conjunct (narrow scope) or it is a dependent of the entire coordination (wide scope).

There is a general ambiguity in the English language regarding the scope of premodification. The following sentence is an example where such a genuine ambiguity exists:

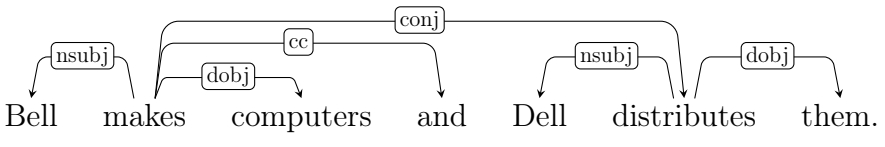


(A.6)

It is not clear if "*good*" modifies both "*food and drinks*", or it modifies "*food*" only. However, there are also sentences which are not ambiguous, but SD is nonetheless incapable of specifying the scope. For example, consider the following two sentences:



(A.7)



(A.8)

"*Bell*" is the subject of both "*makes and distributes*" in the first sentence, but is only the subject of "*makes*" in the second sentence. This difference is not reflected in SD, as the dependency regarding "*Bell*" is the same in two dependency trees: `nsubj(makes, Bell)`.

A.3.1 Resolving scope ambiguity

Making propositions, however, requires resolving the scope ambiguity. In order to represent different scopes, during preprocessing I transform a dependency tree in

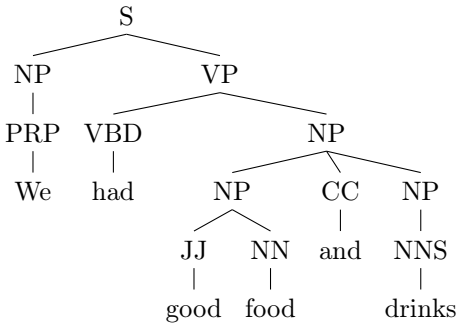
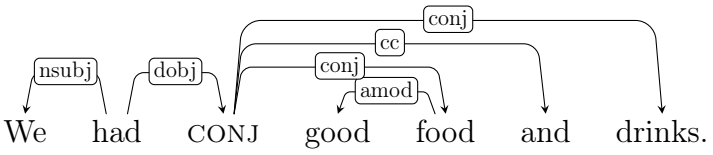


Figure A.2: Constituency parse tree that involves coordination

such a way that coordination is represented by a virtual node (denoted by CONJ in this thesis), which the dependencies of the coordination should attach to. The first conjunct should therefore only have narrow-scope dependencies. A possible result of transforming the dependency tree A.6 is the following:



(A.9)

I transform the dependency tree in the following steps. First, the incoming edge of the coordination head is changed to connect to the virtual node CONJ. All conjuncts and the coordinator then become children of CONJ. Now, every outgoing edge of the first conjunct needs to be classified into either narrow or wide scope. In this example, there is only one edge that needs classification: `amod(food, good)`. If the scope is determined to be narrow, the edge should remain under the first conjunct; if the scope is wide, the edge should be changed to attach under the CONJ node.

The classification of dependencies is done using the constituency parse tree. First, I find the minimum subtree that contains both the first conjunct and the dependent in question. The parse tree of this example is shown in figure A.2. The first conjunct is "*food*"; the dependent in question is "*good*". Therefore the minimum subtree is a noun phrase consisting only of these two words. Then, if any of the other conjuncts is outside the minimum subtree, the scope of the dependent is determined to be narrow, which is the case here because "*drinks*" falls outside the minimum subtree. If the minimum subtree covers all the conjuncts, however,

two heuristics are used in succession to determine the scope:

Position heuristic: If the first conjunct and the dependent in question are separated in text by any other conjunct, the scope of the dependent is wide. If the dependent is situated between the first conjunct and all the other conjuncts, the scope is narrow. After applying the position heuristic, the only remaining case is when the dependent precedes all conjuncts including the first conjunct, which is handled by the following heuristic.

Type heuristic: If any other conjunct has a dependency of the same type as the dependency in question, the scope of the dependency is narrow. Otherwise, the scope is wide. Here, similar dependency types that have complimentary distributions, such as nominal subject (`nsubj`) and clausal subject (`csubj`), are considered to be the same. This is based on the intuition that conjuncts are likely to demonstrate syntactic parallelism (Charniak and Johnson, 2005; Hogan, 2007). For example, using this heuristic alone (hypothetically sidestepping the parse tree test), we can conclude that "*Bell*" has a wide scope in sentence A.7 but a narrow scope in sentence A.8, because in the latter case the other conjunct "*distributes*" has its own subject "*Dell*".

A.3.2 Distributing coordinated arguments/predicates

During the tree traversal, I create a coordination proposition for each CONJ node. A coordination proposition is always predicated by a CONJ node, or a quantifier of coordination (`preconj` in SD, such as "*both*" and "*either*") if one is present. I store coordinators as the label of the coordination predicate. When creating other propositions, a CONJ node is regarded as a regular node. For example, the propositions derived from the dependency tree A.9 are as follows:

```
have (we, CONJ)
and:  CONJ (food, drink)          (A.10)
good (food)
```

Note that propositions now may contain CONJ, which represents coordinated arguments or predicates.

During the post-processing of propositions, a proposition can be distributed over the conjuncts in order to create smaller units of meaning. This is in contrast

to KvD, who never distribute a proposition over coordinated arguments, but who nonetheless have the luxury of using human intelligence to construct new propositions.

For a CONJ that acts as an argument of a proposition, I distribute the proposition over the conjoined arguments only if the coordination is a conjunction, which means every conjunction is assumed to be distributive, but other types of coordination (such as disjunction) are assumed non-distributive.[2] This of course is subject to loss of meaning when it comes to sentences that require a collective reading, such as "*Alice and Jeff are married*". If more than one argument is a conjunction, I distribute the proposition over one conjunction at a time, and then apply the method recursively on the resulting propositions until no distribution is required. After processing the propositions A.10, the proposition `have (we, CONJ)` is replaced by the distributed propositions `have (we, food)` and `have (we, drink)`.

If the predicate of a proposition is a CONJ, I create copies of the proposition, each predicated by one of the conjoined predicates, and replace the coordination of predicates with the coordination of propositions. For example, tree traversal of an augmented version of the dependency tree A.7 would result in the following propositions:

```
and:  CONJ (make, distribute)
CONJ (Bell, computer)
```
(A.11)

I thus turn the coordinated predicates into coordinated propositions:

```
#1 = make (Bell, computer)
#2 = distribute (Bell, computer)
#3 = and:  CONJ (#1, #2)
```
(A.12)

A.4 Other types of propositions

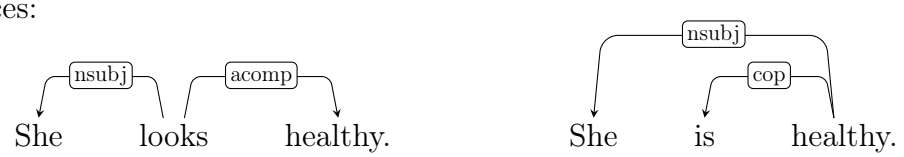
In the previous subsections, I have presented the creation of many important types of propositions. The full picture of my proposition building process, however, consists of a few more considerations for specific dependency types, namely the

[2]It would be also possible to distribute a proposition over a coordinated argument of the form "*A instead of B*" or "*A but not B*", by adding negation to the distributed proposition that involves "*B*". However, this is not currently pursued, due to limited usage of such coordination.

handling of copula and apposition, and of subordinate clauses such as relative clauses and adverbial clauses. In this section, I illustrate my approach to these dependency relations using examples.

A.4.1 Copula propositions

There is a grey area regarding whether a copular verb should become a predicate. In SD, the copula "*be*" is normally regarded as an auxiliary modifier of its complement, which would be designated the predicate over the subject of a sentence. This differs from the treatment of other copular verbs, which would become predicates. For example, compare the dependencies of the following two sentences:



The resulting propositions would be `look (she, healthy)` and `healthy (she)`, respectively. However, because SD applies the special treatment of the copula "*be*" unconditionally, there is a risk that a concept that does not normally have predicative meaning would be forced to be a predicate only because it occurs as the complement of a copula "*be*". For example, consider the following two sentences:

$$\begin{aligned} &\alpha. \quad \textit{The president of the council is Cameron.} \\ &\beta. \quad \textit{Cameron is the president of the council.} \end{aligned} \tag{A.13}$$

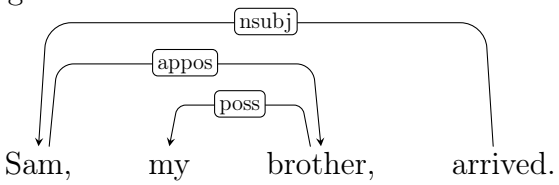
Without adjustment, an awkward proposition `Cameron (president)` would have been made for sentence α; whereas sentence β would have given rise to `president (Cameron)`, which is arguably more natural. The awkwardness comes from the fact that "*Cameron*" does not possess predicative meaning. Because "*president*" can be regarded as a realization of the predicate PRESIDE, an intelligent proposition for either sentence would be PRESIDE (CAMERON, COUNCIL).

I approach copulas in the following way. First, I disable the special treatment of the copula "*be*" of SD, by using the option `-makeCopulaHead` in Stanford parser. Thus, every copula connects to its complement via an `acomp` or `xcomp` edge. Then, during proposition building, I turn the complement of a copula "*be*" into a predicate only if this complement is an adjectival complement (`acomp`), which is the case for sentences such as "*She is healthy*". In all other cases, a copula

remains a predicate, because it is difficult to determine whether the subject or the complement of a copula should become the predicate (or neither can be). The resulting propositions for sentence α in (A.13) are as follows:

```
be (president, Cameron)
of (president, council)
```
(A.14)

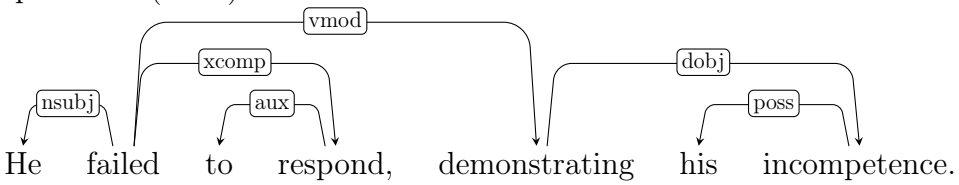
I approach apposition using a similar method. Apposition is represented in SD using edges of relation type `appos`, which connect two noun phrases. For example, consider the following sentence:



Similar to copulas, it is difficult to determine which side of apposition should become the predicate, although in this example "*brother*" is a relational concept, which could serve as a predicate. Therefore, I create a copula proposition by introducing a virtual token `BE` as the predicate and using both ends of the `appos` relation as arguments: `BE (Sam, brother)`. In some rare cases, an adjectival or adverbial modifier is mistakenly labelled as apposition in SD. I identify these cases using part-of-speech information, and treat the modifiers as usual (i.e. as predicates).

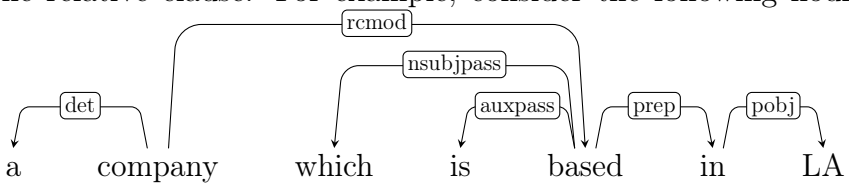
A.4.2 Turning modifier proposition into predicate proposition

Modifiers which are verbs sometimes take arguments in addition to the thing being modified. In order to add additional arguments as they are found, during the tree traversal I sometimes modify a proposition after it is created. For example, consider the following sentence, which contains a verbal modifier in the present participial form (`vmod`):



Two propositions are created at the node "*failed*" during tree traversal: a predicate proposition `fail (he, respond)`, as well as a modifier proposition `demonstrate (fail)`. Then, tree traversal moves on to create propositions for "*demonstrating*", which is a child of the node "*failed*". This time, the direct object is added to the already-created proposition, turning it into a predicate proposition: `demonstrate (fail, incompetence)`. The general rule is that, before creating a new proposition for a predicate, check whether a proposition of this predicate already exists, and if so add the newly found arguments to that proposition instead of creating a new one.

This is also the way I handle relative clauses (`rcmod`), which differs from `vmod` mainly in that I need to replace the relative pronoun with the word modified by the relative clause. For example, consider the following noun phrase:



When I create propositions for "*based*", I add new arguments to the already-created proposition `base (company)`. In this case, the newly found argument is the passive subject "*which*". Because a relative pronoun refers to the only argument of the already-created proposition, it is not an additional argument, but shares the same slot as the existing argument. I record this information by making the relative pronoun a label of the existing argument. Because this proposition has no argument other than the subject, the PP "*in LA*" is also included as an argument (cf. Section A.2). The resulting proposition is `base (which: company, in: LA)`.

A.4.3 Information beyond local context

During tree traversal, sometimes a token that is not in the local context (the current node and its children) is required as an argument of the proposition being built. Two variables are kept track of to provide such information. One of them is the controlling subject (`xsubj`), which would take over the role of subject if the proposition under focus does not have a subject. For example, consider the

propositions of the predicate "*eat*" and "*respond*" in the following two sentences:

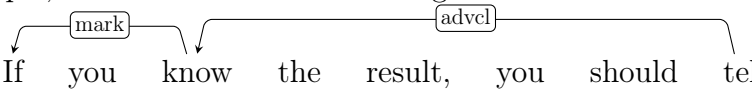
$$\begin{array}{ll} \alpha. & \textit{John likes to eat fish.} \\ \beta. & \textit{Sue asked George to respond to her letter.} \end{array} \tag{A.15}$$

In both cases, a subject cannot be found within the infinitive clause, and must be inherited from higher clauses. If a node has an object as its child, the object is the controlling subject when creating propositions for its children. This is the case in β, in which the object "*George*" becomes the subject of "*respond*": `respond (George, to: letter)`. If a node does not have an object, the current subject is passed on as the controlling subject, as is the case in α: `eat (John, fish)`.

The other variable is the main proposition of a sentence, i.e. the predicate proposition created for the root node. This is useful when the current sentence itself is in coordination with the previous context, which is assumed to be represented by the main proposition of the previous sentence. In SD, inter-sentential coordination is indicated by a `cc` dependent of the root node of a sentence. For example, consider the sentence "*However, she didn't complain*", which has a dependency `cc(complain, However)`. This type of `cc` can be distinguished from the ones I discussed in Section A.3, because it does not co-occur with `conj` relations. Using the coordinator as the predicate, a proposition shall be created to connect the main propositions of the previous and the current sentences. For example, suppose that the previous sentence is "*The food was awful*", the connective proposition would be `however (awful (food), not complain (she))`.

A.4.4 Connective propositions

Within a sentence, connective propositions that express relations such as causality, purpose, condition, and time circumstance can appear as adverbial clause (`advcl`) relations in SD. The marker (`mark`) introducing the subordinate clause is the relation word that should be used as the predicate of the connective proposition. For example, the `advcl` and the `mark` edges are shown in the following sentence:



When I create propositions for "*tell*", the grandchild "*If*" is used to create the proposition `if (should tell, know)`, which would later become `if (should`

Relation types	Status in propositions
mwe, nn, number, prt, goeswith	Eliminated (merged with parent)
aux, det, predet, neg, num, quantmod, expl, discourse	Eliminated (joined to parent)
arg, conj	Argument
acomp	Argument if parent node is not copula "*be*", otherwise predicate over the subject of parent node
amod, advmod, vmod, rcmod	Predicate
prep	Classified as either predicate or the label of argument (pobj or pcomp being the argument)
cc	Label of coordination predicate, or predicate of connective proposition (when without conj)
preconj	Predicate of coordination proposition
appos	Argument of copula proposition
poss	Argument of possessive proposition, or predicate (when without possessive)
possessive	Predicate of possessive proposition
advcl, parataxis, tmod	Argument of connective proposition
npadvmod	Argument if parent node is a predicate, otherwise argument of connective proposition
mark	Predicate of connective proposition

Table A.1: Outcome of dependencies grouped by relation types

tell (you, me), know (you, result)) by creating embedded propositions. If a mark relation is not found, I look for markers which are often mislabelled as advmod, such as "*when*". The virtual token RELATED is used as a back-off predicate, if no markers can be found. I treat parataxis (i.e. two sentential phrases placed side by side without explicit coordination or subordination) in the same way, except that the predicate is always a virtual node because no relation information is given.

A.5 Summary of rules

I have summarized the action to take for each dependency type of SD in table A.1. In this table, a type includes its subtypes, except for subtypes that are also listed; for instance arg includes nsubj and many others, but not acomp. In the

status column, “argument” without specification means argument of its parent node; “predicate” without specification means predicate over its parent node. The definition of connective proposition follows KvD (cf. page 159), but excludes coordination propositions. A connective proposition with an empty predicate (represented by the virtual token `RELATED`) is also the back-off strategy for relations that the parser fails to classify (i.e. relation type `dep`).

Appendix B

Additional pseudocode

To facilitate reimplementation, this appendix provides additional algorithmic descriptions of the subroutines that are described in Section 3.3.

Algorithm 7 The method used in the first memory cycle to determine the initial root of the memory tree. It is described in Subsection 3.3.2.1, which contains a tie-breaking mechanism for Line 14. It is used by Algorithm 1.

Require: *propositions*, a set of propositions
Ensure: a graph connecting *propositions* is returned
1: **function** MakeAuxiliaryGraph(*propositions*) ▷ helper function
2: $G \leftarrow$ an empty bidirected graph
3: **for all** $p \in pendingPropositions$ **do**
4: **for all** $q \in pendingPropositions \wedge q \neq p$ **do**
5: **if** $overlap(q, p) > 0$ **then**
6: create edge in G from p to q of weight $\frac{1}{overlap(q,p)}$
7: **end if**
8: **end for**
9: **end for**
10: **return** G
11: **end function**
Require: *pendingPropositions*, a set of propositions
Require: *tree*, an empty memory tree
Ensure: one proposition in *pendingPropositions* becomes the root of *tree*, and is removed from *pendingPropositions*
12: **procedure** InsertRoot(*pendingPropositions*, *tree*)
13: $G \leftarrow$ MakeAuxiliaryGraph(*pendingPropositions*)
14: $root \leftarrow \text{argmax}_p\, closenessCentrality(G, p)$
15: add *root* to *tree* as its root
16: remove *root* from *pendingPropositions*
17: **end procedure**

Algorithm 8 The method used in every cycle to reset the memory tree if necessary. It is described in Subsection 3.3.2.1, and is used by Algorithm 1. The helper function MakeAuxiliaryGraph is defined in Algorithm 7.

Require: *pendingPropositions*, a set of propositions
Require: *tree*, a memory tree
Ensure: the *tree* is reset using one of *pendingPropositions* as the root if doing so can make more propositions connected

1: **function** StartAlternativeTree(*pendingPropositions*, *tree*)
2: **if** *tree* has more nodes than *pendingPropositions* **then**
3: **return** false
4: **end if**
5: $G \leftarrow$ MakeAuxiliaryGraph(*pendingPropositions*)
6: $G' \leftarrow$ the largest connected component of G
7: **if** G' has more nodes than *tree* **then**
8: $root \leftarrow \text{argmax}_p\, closenessCentrality(G', p)$
9: remove all nodes from *tree* and add them to *pendingPropositions*
10: add *root* to *tree* as its root
11: remove *root* from *pendingPropositions*
12: **return** true
13: **else**
14: **return** false
15: **end if**
16: **end function**

Algorithm 9 The method to adjust the root of memory tree according to Subsection 3.3.2.2. It is used by Algorithm 1, where it may be called repeatedly until no change is necessary.

Require: *tree*, a memory tree

Ensure: make a child of the current root of *tree* the new root and return true, if the subtree rooted at it is large enough, otherwise return false

function AdjustRoot(*tree*)
    $v \leftarrow$ the root of *tree*
    for all $u \in$ children of v **do**
        $weight_u \leftarrow \sum_{p \in \text{ subtree rooted at } u} w_p$
    end for
    $totalWeight \leftarrow \sum_u weight_u$
    for all $u \in$ children of v **do**
        $score_u \leftarrow overlap(v, u) \cdot weight_u - overlap(u, v) \cdot (totalWeight - weight_u)$
    end for
    $candidate \leftarrow \text{argmax}_u\, score_u$
    if $score_{candidate} > 0$ **then**
        invert the edge from v to *candidate*, i.e. *candidate* is now the root of *tree*
        return true
    else
        return false
    end if
end function

Algorithm 10 The method to select nodes to retain in working memory at the end of a memory cycle. It is described in Subsection 3.3.2.3, which also defines the notion of recency. It is used by Algorithm 1. The iterator TraverseActivatedNodes(*tree*) yields the activated nodes of *tree* in breadth-first order, and in descending order of recency for nodes of the same depth.

Require: *tree*, a memory tree
Ensure: only a small number of the nodes of *tree* are activated, the others are deactivated

```
 1: procedure MemorySelect(tree)
 2:     retained ← ∅
 3:     quota ← the memory size prescribed by the model
 4:     p ← the root of tree
 5:     while |retained| < quota do
 6:         retained ← retained ∪ {p}
 7:         if p has at least one activated child then
 8:             p′ ← the most recent activated child of p
 9:             if recency_p′ > recency_p then
10:                 p ← p′
11:             else
12:                 break
13:             end if
14:         else
15:             break
16:         end if
17:     end while
18:     for all p ∈ TraverseActivatedNodes(tree) do
19:         if |retained| < quota then
20:             retained ← retained ∪ {p}
21:         else
22:             break
23:         end if
24:     end for
25:     deactivate all nodes of tree that are not in retained
26: end procedure
```

Appendix C

Corpus statistics

The details about the texts and summaries used in the experiments (Section 5.2) are presented in the following table.

- The first four columns contain information about each source text. Texts which do not have titles are given self-made titles here (marked with asterisks). #S and #W stand for number of sentences and number of words, respectively. The FK column contains the Flesch–Kincaid grade level of the source texts.
- The last four columns contain averaged statistics across the four human-written summaries of each source text. As is the case before, #S is the average number of sentences per summary. The average unigram/bigram/trigram overlap percentages between the summaries and the source text are listed in columns 1G, 2G, and 3G, respectively.

Source text				**Summaries**			
Title	#S	#W	FK	#S	1G	2G	3G
A chronicle of timekeeping	36	906	13.4	6.8	78	35	17
A remarkable beetle	35	789	12.7	5.3	85	52	34
A spark, a flint: How fire leapt to life	35	732	12.4	7.3	80	41	24
Air traffic control in the USA	37	882	13.6	7.0	88	62	47
Airports on water	57	938	8.4	6.0	85	46	28
Architecture – reaching for the sky	40	764	13.0	5.3	76	41	26
Attitude to language	29	609	12.9	6.0	77	40	26
Cinema changed the world*	47	776	9.7	5.8	73	26	10

Early childhood education	45	943	12.4	5.8	79	42	27
Effects of noise	31	729	12.4	7.0	88	49	29
In search of the holy grail	42	996	13.0	5.3	78	47	32
Language barriers*	30	742	14.2	6.3	84	49	31
Lost for words	56	864	10.2	6.5	75	35	18
Making every drop count	34	816	14.4	6.5	78	48	33
Motivating employees under adverse condition	46	903	13.1	7.0	82	36	18
Numeration	36	896	12.6	6.5	81	39	19
Obtaining linguistic data	43	896	13.2	7.3	83	47	32
Pulling string to build pyramids	47	803	9.3	7.0	77	33	18
Spoken corpus comes to life	31	585	11.1	5.0	67	22	10
The birth of scientific English	40	900	14.3	6.8	79	45	21
The concept of role theory	44	964	11.1	6.8	82	46	29
The department of ethnography	34	788	15.2	5.0	85	55	37
The development of museums	39	898	14.2	7.0	84	57	40
The motor car	35	673	11.5	5.8	69	24	11
The nature of genius	31	1011	16.6	5.8	81	44	32
Tidal power	32	771	12.7	4.8	78	41	24
Tourism	31	767	15.0	5.5	82	41	25
Visual symbols and the blind	40	746	10.2	5.0	76	33	13
Volcanoes – earth-shattering news	56	1009	9.4	7.8	83	45	22
Votes for women	30	826	14.6	4.5	67	36	19
What's so funny? John McCrone reviews recent research on humour	51	884	11.0	6.0	73	30	12
Average	39	832	12.5	6.1	79	42	25

Appendix D

Example summaries

This appendix contains the outputs of most of the summarizers evaluated in Chapter 5 for document *Pulling string to build pyramids*. The colours of the names of the systems indicate the system's category in Section 5.4.

The corresponding human-written reference summaries have been shown in Table 5.4. The outputs of two abstractive systems, PbAbs and Seq2seq, have also been shown as an example of the pyramid evaluation in Table 5.6. The outputs of all other systems in Section 5.4, and the most representative systems in Section 5.3, are listed below. The outputs of the pipeline systems are shown immediately after the extractive systems they are based on, so that it is easy to notice the effect of sentence compression (sentence compression also makes room for including additional sentences).

PbExt: She wondered if perhaps the bird was actually a giant kite, and the men were using it to lift a heavy object. He says. And since he needed a summer project for his student Emilio Graff, investigating the possibility of using kites as heavy lifters seemed like a good idea. Gharib and Graff set themselves the task of raising a 4.5-metre stone column from horizontal to vertical, using no source of energy except the wind. So Clemmons was right: the pyramid builders could have used kites to lift massive stones into place. The experiments might even have practical uses nowadays.

PbExt + Clarke and Lapata: The pyramids of Egypt were built years and no one knows how. She wondered if the bird was a kite and men were using it lift object. Coming I have a keen interest in Middle Eastern science, he says. Since he needed a summer project investigating the possibility of using kites seemed like a good

idea. Gharib set themselves the task of raising stone column using source of energy except the wind. Year team put Clemmons's theory using sail. So Clemmons was right pyramid builders have used kites lift stones into place. The evidence is non-existent. The experiments have uses.

PbExt + Cohn and Lapata: They were holding what looked like ropes. Perhaps the bird was, and the men were a heavy object. 'Coming in iran, i have a keen interest,' . And since he needed a summer project for his student, investigating the possibility seemed like a good idea. Gharib and graff set themselves the task to vertical, using no source. Earlier this year the team. The kite to clean off the ground. The pyramid builders could used kites lift massive stones. The evidence for using is no method. The evidence the kite-lifting says The experiments might even have practical uses. There are places.

LexRank: Their initial calculations and scale-model wind-tunnel experiments convinced them they wouldn't need a strong wind to lift the 33.5-tonne column. Even a 300-tonne column could have been lifted to the vertical with 40 or so men and four or five sails. So Clemmons was right: the pyramid builders could have used kites to lift massive stones into place. There are no pictures showing the construction of the pyramids, so there. In addition, there is some physical evidence that the ancient Egyptians were interested in flight. His idea is to build the arches horizontally, then lift them into place using kites.

LexRank + Clarke and Lapata: She wondered if the bird was a kite and men were using it lift object. Their calculations and experiments convinced them they wouldn't need wind lift column. Kite lifted column clean off the ground. Column have been lifted with men. So Clemmons was right pyramid builders have used kites lift stones into place. There are pictures showing the construction of the pyramids, there is no way to tell what happened. The evidence is non-existent. In addition, there is evidence that the Egyptians were interested in flight. His idea is to build arches lift them into place using kites.

LexRank + Cohn and Lapata: Perhaps the bird was, and the men were a heavy object. He was by the picture that. Their initial calculations convinced they wouldn't need a strong wind. The kite to clean off the ground. The instant the sail a force. He could have lifted to the vertical. The pyramid builders could used kites lift massive stones. There are pictures, so to tell what really happened. The evidence the kite-lifting says. There is some physical evidence that were interested

in flight. Its sophistication suggests might developing ideas for a long time. His idea is to build the arches to using kites.

PbTok: The pyramids of Egypt were built, and knows. They were holding looked that led, via some kind of system, to a bird. The men were using it to lift a object. Intrigued, Clemmons contacted Morteza Gharib, Have' he says. The possibility of using kites as lifters. Gharib set the task of raising, using no source except the wind. Put, using a nylon sail. The kite lifted the column off the ground. Could have used to lift stones into place. The evidence for using, Sailors like the Egyptians. They are known to have used, BC, were using. The experiments might have uses.

MEAD: The pyramids of Egypt were built more than three thousand years ago, and no one knows how. She wondered if perhaps the bird was actually a giant kite, and the men were using it to lift a heavy object. Their initial calculations and scale-model wind-tunnel experiments convinced them they wouldn't need a strong wind to lift the 33.5-tonne column. The instant the sail opened into the wind, a huge force was generated and the column was raised to the vertical in a mere 40 seconds. ' The evidence for using kites to move large stones is no better or worse.

TextRank: She wondered if perhaps the bird was actually a giant kite, and the men were using it to lift a heavy object. Gharib and Graff set themselves the task of raising a 4.5-metre. This jerk meant that kites could lift huge weights, Gharib realised. So Clemmons was right: the pyramid builders could have used kites to lift massive stones into place. The evidence for using kites to move large stones is no better or worse than the evidence for the brute force method, Gharib says. His idea is to build the arches horizontally, then lift them into place using kites.

SummaRuNNer: The pyramids of egypt were built more than three thousand years ago, and no one knows how. The conventional picture is that tens of thousands of slaves dragged stones on sledges. But there is no evidence to back this up. Now a californian software consultant called maureen clemmons has suggested that kites might have. She wondered if perhaps the bird was actually a giant kite, and the men were using it to lift a heavy object. Coming from iran, i have a keen interest in middle eastern science,' he says. He says. Indeed, the experiments have left many specialists unconvinced.'