5-11-2010

# Methods for Sentence Compression

Emily Pitler
*University of Pennsylvania*

# Methods for Sentence Compression

**Abstract**
Sentence compression is the task of producing a summary of a single sentence. The compressed sentence should be shorter, contain the important content from the original, and itself be grammatical. The three papers discussed here take different approaches to identifying important content, determining which sentences are grammatical, and jointly optimizing these objectives. One family of approaches we will discuss is those that are *tree-based*, which create a compressed sentence by making edits to the syntactic tree of the original sentence. A second type of approach is *sentence-based*, which generates strings directly. Orthogonal to either of these two approaches is whether sentences are treated in isolation or if the surrounding discourse affects compressions. We compare a tree-based, a sentence-based, and a discourse-based approach and conclude with ideas for future work in this area.

**Comments**
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-20.

# Methods for Sentence Compression

Emily Pitler

May 11, 2010

**Abstract**

Sentence compression is the task of producing a summary of a single sentence. The compressed sentence should be shorter, contain the important content from the original, and itself be grammatical. The three papers discussed here take different approaches to identifying important content, determining which sentences are grammatical, and jointly optimizing these objectives. One family of approaches we will discuss is those that are *tree-based*, which create a compressed sentence by making edits to the syntactic tree of the original sentence. A second type of approach is *sentence-based*, which generates strings directly. Orthogonal to either of these two approaches is whether sentences are treated in isolation or if the surrounding discourse affects compressions. We compare a tree-based, a sentence-based, and a discourse-based approach and conclude with ideas for future work in this area.

## 1 Introduction

Most summarization systems today produce summaries by identifying the most information in the input and then including the sentences that contain these facts in the summary. Including irrelevant details simply because they happened to appear in the same sentence as something important is a waste of the limited space available for a summary. *Sentence compression* is therefore an important problem for automatic summarization systems (Jing, 2000).

Besides summarization, there are several other real applications in which the ability to produce compressed sentences is beneficial. Examples include: compressing sentences so television closed captions can keep pace with the program (Linke-Ellis, 1999; Robert-Ribes et al., 1999), compressing sentences before indexing in information retrieval (Corston-Oliver and Dolan, 1999), and incorporating sentence compression into reading machines so that the blind can skim through documents (Grefenstette, 1998).

We define the sentence compression task as follows: given an input sentence, a system must produce a sentence which

- is shorter
- retains the important information from the original
- is grammatical

The task is quite similar to summarization, except on the level of sentences, not full articles. Sentence compression has been described as a "scaled down version of the text summarization problem" (Knight and Marcu, 2002).

Of the three desiderata listed above, length is straight-forwardly quantified as the number of words in the compressed sentence. For importance and grammaticality, however, there are multiple plausible ways to automatically measure these values. The papers we survey differ in how they represent and quantify importance and grammaticality, as well as how they jointly optimize these criteria. Some use pairs of sentences and their compressions to learn which compression operations yield grammatical outputs; others take an unsupervised approach and learn grammaticality from large amounts of text. Similarly, importance can be learned from the types of edits found in the pairs of text and their compressions, or linguistic principles can provide constraints on what must be kept.

We will begin in Section 2 with a description of the work of Knight and Marcu (2002), whose experimental set-up the rest of the field has followed and compared themselves against. There are two models in Knight and Marcu (2002), and both are what we will call in this work *tree-based* approaches–the most probable compressed *syntactic tree* is found, and then the sentence produced is simply the yield of this tree. We discuss the tree-based approach of Galley and McKeown (2007) in Section 3, who improve upon the work of Knight and Marcu (2002) by estimating the probabilities of these trees in different ways. We then introduce what we will call the *sentence-based* approach of McDonald (2006) in Section 4. Rather than using tree transformations, McDonald (2006) finds the highest scoring compressed sentence directly. In addition, McDonald (2006) uses discriminative learning to include features from a variety of sources, including both constituency and dependency trees. The third paper we will examine in detail is that of Clarke and Lapata (2007) in Section 5. Clarke and Lapata (2007) also use a sentence-based approach, but incorporate information from the surrounding *discourse*, rather than compressing sentences in isolation. The method in Clarke and Lapata (2007) is unsupervised, and finds the most probable sentence according to a language model, rather than using compressed and uncompressed sentence pairs as supervised training data. Finally, we compare the three approaches in Section 6 and discuss some opportunities for future work in Section 7.

## 2 Task Set-up: Knight and Marcu, 2002

Knight and Marcu (2002)'s seminal work in sentence compression created a standard corpus for this task, an evaluation set-up, and the standards for future research to compare against.

### 2.1 Data

A practical issue for sentence compression is: how should one create examples? Knight and Marcu (2002) avoid asking humans to manually compress sentences. Instead, they automatically extract pairs of sentences and their compressed versions from full articles and their abstracts. They use the Ziff-Davis corpus, which is a collection of newspaper articles about technological products in which each article has an abstract. The compressed sentences come from these abstracts, and a pair is added if an abstract sentence is a subsequence of words in a sentence in the full article, i.e., one can transform the article's sentence into the abstract sentence by deleting words. From the more than 4,000 articles in the Ziff-Davis corpus, they extract 1067 sentence pairs. This small proportion is probably in part due to the fact that even when writers of abstracts reuse a sentence from the article, they do not just drop words. Human writers include paraphrases or synonyms ("said" versus "stated") and use alternative syntactic constructions ("gave John the book" versus "gave the book to John"). Neither of these alternations would be detected by Knight and Marcu, and so a) are left out of the training and test sets, and b) are not capable of being generated by Knight and Marcu (2002) and most sentence compression algorithms subsequent to them.

## 2.2 Methods

The task is then defined as: given a long sentence $l$, produce a compressed version $c$ which both preserves the meaning of $l$ and is also grammatical.

Knight and Marcu (2002) present two alternative methods for learning to generate these compressed sentences, the *noisy channel* method and the *decision tree* method. Both methods operate directly on syntactic parse trees.

The noisy channel model is generative and has the following generative story: assume that someone, such as a newspaper editor, produces a short sentence $c$, gives it to a reporter and then tells the reporter to add some more details. At test time, Knight and Marcu are given the long sentence $l$, and want to find the most likely short original sentence $c$. $p(c)$ is the source model (prior probability the editor would produce this short sentence) and $p(l|c)$ is the channel model (likelihood of the reporter transforming $c$ into $l$).

The best compression is then found by maximizing the following:

$$p(l, c) = p(c)p(l|c)$$

Note that according to this story, the reporter may add words, but not permute or delete any of the original words. Therefore under this set-up, the set $C$ of possible compressions is the set of sentences which can be derived from $l$ by deleting zero or more words. For a sentence with $n$ words, there are then $2^n$ possible compressions (each word can either be kept or deleted). While this is the set-up for the majority of later sentence compression research, there is some work that considers insertions, substitutions and rearrangements (Cohn and Lapata, 2008).

The channel model requires an estimate of $p(l|c)$. To estimate this, they compute the probability of all of the expansion operations which would be required to convert the parse tree of $c$ ($t_c$) into the parse tree of $l$ ($t_l$). The probability of each of these operations is estimated using the frequencies of these operations in the training set.

They define the source model of their compressed sentence, $p(c)$, as the product of the probability of $t_c$ (estimated based on the frequency of the productions in the tree in the Penn Treebank) and the probability of the bigrams in $c$ (estimated from the words in the Penn Treebank). Each word therefore contributes twice to the probability estimate of the sentence: once in which the presence of the word is conditioned on its parent in the tree (part-of-speech tag) and once in which it is conditioned on the previous word. This source model is not a well-formed probability distribution, as the sum of all compressed sentences would not sum to 1.

Subsequent work has built upon the noisy channel model, and so that is what we have focused on in this survey. However, there is another model included in Knight and Marcu (2002), which they call the decision tree model. The decision tree method learns a decision tree to incrementally convert between original parse trees and compressed parse trees. All of the original words are placed on a stack and then the model learns when it should *shift* and when it should *reduce*, based on features of what is left on the stack and the partially completed portion of the compressed tree.

### 2.3 Evaluation

Knight and Marcu (2002)'s evaluation methodology has been used in almost every sentence compression paper since then. On a held-out set of 32 sentences from the Ziff-Davis corpus, human raters are presented with both the compression produced by the system and the human-written corresponding sentence from the abstract. The human raters are then asked to evaluate each compression on a scale of 1-5 for the following two aspects:

1. Grammaticality
2. Importance ("how well the system did with respect to selecting the most important words in the original sentence" (Knight and Marcu, 2002))

In addition to the two human-produced scores, they also report the *compression rate*, which is the percentage of words in the original sentence that are left in the compressed sentence.[1] The human-produced sentences (sentences from the abstract) had a compression rate of 53%, while the noisy channel model and decision-based model produced longer sentences (compression rates of 70% and 57%, respectively). Since Knight and Marcu (2002), researchers have reported grammaticality, importance, and compression rates, usually on the same test set of the 32 sentences.

They found that the noisy channel model and the decision tree model performed similarly for the Ziff-Davis test set.

## 3 Tree Operations for Sentence Compression: Galley and McKeown, 2007

Given the requirement that compression sentences be grammatical, one approach to sentence compression is to perform operations on syntactic trees directly (Knight and Marcu, 2002; Turner and Charniak, 2005; Galley and McKeown, 2007). The probabilities of tree operations are often estimated using *probabilistic synchronous context-free grammars*. We will review synchronous context-free grammars in section 3.1. In section 3.2, we introduce the work of Galley and McKeown (2007), who improve upon simple synchronous CFGs by both including more powerful rules and by using various techniques developed for parsing to better estimate the rule probabilities.

### 3.1 Synchronous Context-free Grammars

A context-free grammar (CFG) $G$ consists of $(V, \Sigma, P, S)$, where $V$ is the set of non-terminals, $\Sigma$ the set of terminals, $P$ the finite set of production rules, and $S$ the start symbol.

In a CFG, the production rules are rewrite rules of the form:

$$A \to \alpha$$

where $A \in V$ and $\alpha \in (\Sigma \cup V)^*$.

*Synchronous CFGs*, also known as *syntax-directed transduction grammars* (Lewis and Stearns, 1968) or *syntax directed translation schemata* (Aho and Ullman, 1969), include an additional alphabet $\Delta$ and have rules of the form:

$$A \to (\alpha, \beta, \Pi)$$

[1]This definition of compression rate used by Knight and Marcu (2002) and subsequent work corresponds to residual length. A *high* compression rate corresponds to *longer* sentences/*less* compression.

where $A \in V$, $\alpha \in (\Sigma \cup V)^*$, and $\beta \in (\Delta \cup V)^*$. $\Pi$ is a permutation recording a 1-to-1 correspondence between each variable in $\alpha$ and each variable in $\beta$.

Because of the 1-to-1 correspondence required in each production rule, the parse tree for the source sentence and the parse tree for the target sentence are isomorphic. Nodes can be relabeled and daughters can be reordered, but every non-terminal will have the same parent and children in both trees.

The sentence compression task, however, requires *deleting* elements. Synchronous CFGs can express deletion by introducing non-terminals which, on the compressed side, only yield the empty string (Chiang, 2006). For example, we could use the non-terminal $PP$ when a prepositional phrase will be present on both sides, but the non-terminal $PP'$ when a prepositional phrase will be present on the long side but not on the compressed side. Therefore, the type of deletion that can be modeled by a synchronous CFG is the case in which syntactic constituents (which correspond to non-terminals) are dropped.

For example, consider a simple synchronous CFG designed for sentence compression, where the first right-hand side is used for the derivation of the full sentence and the second right-hand side is used for the derivation of the compressed sentence. $V = \{S, NP, VP, Adv, Adv', V, N, N', PP, PP', P, P'\}$, $\Sigma = \Delta = \{I, really, like, spaghetti, with, sauce\}$, $S = S$, and production rules $P$ below. The correspondences necessary between the two right-hand sides of each rule are indicated with subscripts:

$S \rightarrow (NP_1\ VP_2,\ NP_1\ VP_2)$
$NP \rightarrow (I_1,\ I_1)$
$NP \rightarrow (N_1\ PP_2,\ N_1\ PP_2)$
$^*NP \rightarrow (N_1\ PP'_2,\ N_1\ PP'_2)$
$N \rightarrow (spaghetti_1,\ spaghetti_1)$
$N' \rightarrow (sauce_1,\ \epsilon_1)$
$VP \rightarrow (Adv'_1\ V_2\ NP_3,\ Adv'_1\ V_2\ NP_3)$
$V \rightarrow (like_1,\ like_1)$
$Adv' \rightarrow (really_1,\ \epsilon_1)$
$PP' \rightarrow (P'_1\ NP'_2,\ P'_1\ NP'_2)$
$P' \rightarrow (with_1,\ \epsilon_1)$

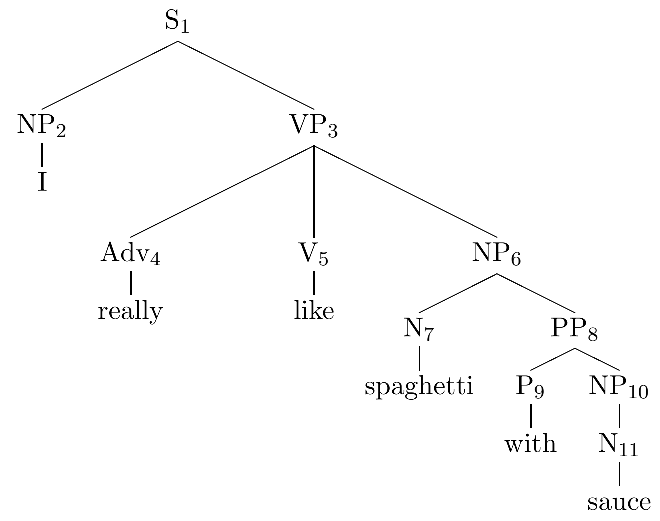
A prepositional phrase is deleted by using the starred rule, rather than the one above it. The use of the non-terminal $PP'$ indicates that the prepositional phrase has been deleted.

From this synchronous CFG, we can derive the long sentence $l$ = *I really like spaghetti with sauce* and a compressed version $c$ = *I like spaghetti.* If these sentences were parsed separately, then the parse tree for $l$ would be figure 1(a), and the tree for $c$ would be the one in figure 1(b). If instead we derive these sentences using the synchronous CFG rules above, the trees produced are figures 1(c) and 1(d). Aligned non-terminals are indicated with subscripts. It is easy to see that the trees in Figures 1(a) and 1(d) are isomorphic to each other. For the remainder of the paper, we will draw compressed trees without the null nodes, as in Figure 1(b), but remember that each is equivalent to a tree with the empty nodes explicitly shown (Figure 1(d)).

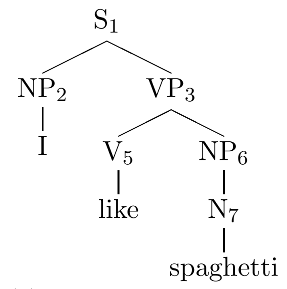
Not all sentence compressions can be modeled exclusively by deleting constituents, however; this will be discussed further in section 3.2.4.

## 3.2 Galley and McKeown, 2007

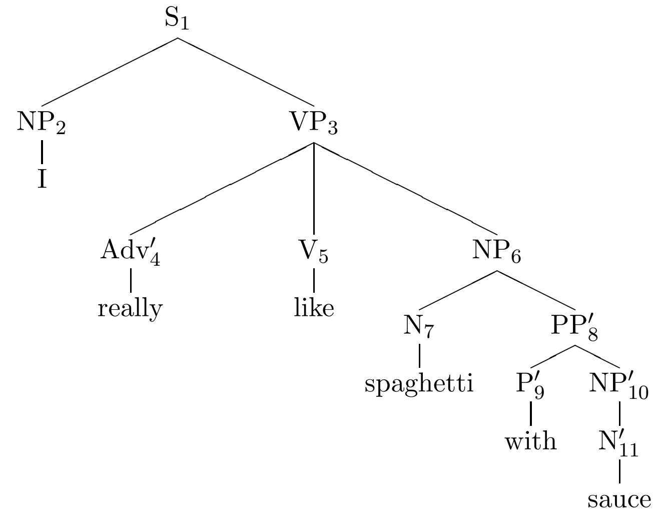
The channel model of Knight and Marcu (2002) estimates the probabilities of synchronous CFG rules. There are two main problems with learning a synchronous CFG for sentence compression:



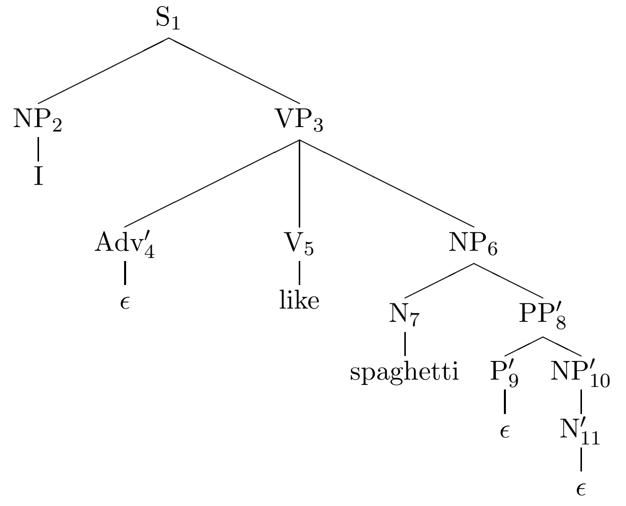
(a) Long sentence parse



(b) Compressed sentence parse



(c) Derivation of the long sentence. This tree is strongly equivalent to the tree in (a).



(d) Derivation of the compressed sentence. This tree is weakly equivalent to the tree in (b).

Figure 1: Alignments of compressed and uncompressed versions. All nodes can be aligned, so these trees can be produced with a synchronous CFG.

1. Many rules are only seen once, and so have unreliable probability estimates
2. Not all pairs of trees can be derived using synchronous CFG rules

Galley and McKeown (2007) improve upon Knight and Marcu (2002)'s noisy channel model in both of these respects. We will first discuss Galley and McKeown (2007)'s use of lexicalization, Markovization, and parent annotation to improve the probability estimates of each rule, and then discuss how Galley and McKeown (2007) and others have dealt with learning more powerful rules.

Both the noisy channel model and Galley and McKeown (2007) are tree-based and find the best compressed tree $\hat{t_c}$ of a long sentence's parse tree $t_l$ by maximizing:

$$\hat{t_c} = \text{argmax}_{t_c}\ p(t_l, t_c)$$

The noisy channel model uses Bayes' Rule and treats $t_c$ as the true variable and $t_l$ as the observed variable:

$$\text{argmax}_{t_c}\ p(t_l|t_c)p(t_c) \tag{1}$$

Galley and McKeown use the chain rule in the opposite direction:

$$\begin{aligned}&\text{argmax}_{t_c}\ p(t_c|t_l)p(t_l)\\ &= \text{argmax}_{t_c}\ p(t_c|t_l)\end{aligned} \tag{2}$$

The $p(t_l)$ term can be dropped since it is constant for all values of $t_c$.

The noisy channel model of Knight and Marcu (2002) finds the likelihood of a long sentence given a compressed sentence (equation 1), while Galley and McKeown (2007) instead compute the likelihood of a compressed sentence given a long sentence (equation 2). One advantage to maximizing equation 2 directly is that the problematic $p(t_c)$ term is no longer needed. To truly estimate the probability of a particular compressed sentence, one should gather a large corpus of compressed sentences. Instead, Knight and Marcu estimate $p(t_c)$ using a large corpus of *uncompressed* sentences. Turner and Charniak (2005) point out that Knight and Marcu (2002) are forced to assume that the probability of a particular sentence being a compressed sentence is equal to its probability of being a regular sentence, while we would expect qualitative differences between compressed and uncompressed sentences. Turner and Charniak show that this assumption causes the noisy channel model to drastically favor non-compressions.

Since Galley and McKeown (2007) are using probabilistic synchronous CFGs, the probability of two trees can be decomposed into the product of all the production rules used to construct the trees. If $r_1$, $r_2$,...,$r_J$ rules are used, where $r_j$ is the rule $lhs_j \rightarrow (rhs_l, rhs_c)$, then:

$$\text{argmax}_{t_c}\ p(t_c|t_l) = \prod_{j=1}^{J} p(rhs_c|lhs_j, rhs_l) \tag{3}$$

The main contribution of Galley and McKeown (2007) is their analysis of several different ways of computing $p(rhs_c|lhs_j, rhs_l)$ for each production rule.

The most straight-forward way to compute the probability of each synchronous CFG rule is simply by reading off the counts of each time $lhs_j$ was expanded into $rhs_l$ and $rhs_c$ in the aligned trees in the training set, divided by the number of times $lhs_j$ occurred.

However, work in parsing which learns probabilistic CFGs from Penn Treebank (Marcus et al., 1994) style parse trees, has found these probability estimates have been found to be less than ideal (Johnson, 1998; Collins, 1999; Klein and Manning, 2003). Parent annotation (Johnson, 1998), Markovization (Collins, 1999), and lexicalization (Collins, 1999) have all been shown to improve the quality of the rule probability estimates (i.e., on a held-out test set, the maximum likelihood parses according to these rules are higher scoring than the maximum likelihood parses with rule probabilities estimated in the most straight-forward way). Klein and Manning (2003) systematically analyzed the effect of various combinations of parent annotation and Markovization on parsing accuracy.

In a similar vein, Galley and McKeown extend each of these modifications to the case of *synchronous* probabilistic CFGs, and analyzes their effect on *sentence compression* accuracy. We will discuss each of these modifications in turn, compare the sentence compression results to the parsing results, and derive the estimates for $p(rhs_c|lhs_j, rhs_l)$.

### 3.2.1 Markovization

The trees in the Penn Treebank are flat (have a large branching factor). This flatness causes CFGs induced from the Penn Treebank to have production rules with long right-hand sides. These flat production rules cause there to be a large number of rules that occur infrequently. These flat rules also reduce some generalization–for example, rather than learning that adjectives can often be deleted from noun phrases, we would separately learn the probability of adjective deletion in each of the following rules:

- NP → (DT JJ NN, DT NN)
- NP → (DT JJ NN NN, DT NN NN)
- NP → (DT JJ NN PP, NN)

$DT$ is the Penn Treebank notation for determiner, $JJ$ for adjectives, and $NN$ for common nouns.

In parsing, this problem is recognized, and so researchers often assume that the probability of a variable depends on only the previous $s$ (a small constant) siblings to reduce the effect of sparse data and allow for more generalization. This is equivalent to making an $s$-th order *Markov assumption.* To make this assumption, we need to define what we mean by "previous". Rather than assuming that variables are generated left to right, and so a variable depends on the $s$ siblings directly to its left, (Collins, 1999) makes a more linguistically plausible assumption. Linguistically, the *head* of a phrase is assumed to be the most important, so the head is assumed to be generated first, and then modifiers are generated outwards from the head. The probability of a modifier then depends on the head and its $s$ closest siblings in the direction of the head (Collins, 1999).

Now that we have discussed in general how Markovization applies to parsing, let us examine how Galley and McKeown (2007) apply this concept to sentence compression, where we have two distinct right-hand sides.

**Rule Probabilities Without Markovization**: For each $rhs_l$ in each production rule, the head $H$ is identified; all other variables in $rhs_l$ are identified by whether they are on the left or the right of $H$ and their distance from $H$. If there are $m$ modifiers to the left of the head and $n$ modifiers to the right of the head, then:

$$rhs_l = L^m \; ... \; L^1 \; H \; R^1 \; ... \; R^n \tag{4}$$

As Galley and McKeown (2007) restrict themselves to constituent deletion, with no reordering, we know that any possible right-hand side $rhs_c$ is a subsequence of $rhs_l$. Therefore, we can express $rhs_c$ as a sequence of binary variables $k_l^m$,..., $k_l^1, k_h, k_r^1, ..., k_r^n$:

$$p(rhs_c|lhs_j, rhs_l) = p(k_l^m, ..., k_l^1, k_h, k_r^1, ..., k_r^n|lhs_j, rhs_l) \tag{5}$$

where $k_h = 1$ iff the head of $rhs_l$ is kept in $rhs_c$, $k_l^i = 1$ iff the variable $i$ positions to the left of the head is kept, and $k_r^i = 1$ iff the variable $i$ positions to the right of the head is kept. For example, in the rule (NP → (DT JJ NN, DT NN)), the head of the right-hand side of the long sentence is $NN$, and this rule would have $k_l^2 = 1, k_l^1 = 0, k_h = 1$, indicating that the variable 2 positions to the left of the head (DT) and the head itself (NN) of the long right-hand side are kept in the compressed right-hand side.

Using the chain rule and substituting $rhs_l$ with $L^m...H...R^n$ (from equation 4), equation 5 can be rewritten as:

$$\begin{aligned} p(rhs_c|lhs_j, rhs_l) &= p(k_h|lhs_j, L^m, ..., L^1, H, R^1, ..., R^n) \\ &\cdot \prod_{i=1}^{m} p(k_l^i|k_l^{i-1}, ..., k_l^1, k_h, lhs_j, L^m, ..., L^1, H, R^1, ..., R^n) \\ &\cdot \prod_{i=1}^{n} p(k_r^i|k_r^{i-1}, ..., k_r^1, k_h, k_l^m, ..., k_l^1, lhs_j, L^m, ..., L^1, H, R^1, ..., R^n) \end{aligned} \tag{6}$$

**Rule Probabilities With Markovization**: Just as Collins (1999) and Klein and Manning (2003) make an order-$s$ Markov assumption around the head for parsing, Galley and McKeown (2007) make an order-$s$ Markov assumption around the head for sentence compression. For example, consider our estimate of whether or not we should delete the *NP spaghetti with sauce* in our example. The *NP* is to right of the head *V*. Without any Markovization (using just the chain rule decomposition in equation 6), the probability of keeping the NP in the compression is:

$$p(k_r^1|k_h = 1, k_l^1 = 0, lhs_j = VP, L^1 = ADV, H = V, R^1 = NP)$$

Note that in the above equation, the probability depends on absolutely every sibling to its left. With an order-1 Markov assumption around the head, the probability used is instead:

$$p(k_r^1|k_h = 1, lhs_j = VP, H = V, R^1 = NP)$$

We are now no longer splitting our statistics for whether or not an object should be kept based on whether or not an adverb occurred before the verb and whether or not the adverb was kept, so our statistics should be more robust.

In the general case, equation 6 is simplified to:

$$\begin{aligned} p(rhs_c|lhs_j, rhs_l) &= p(k_h|H, lhs_j) \\ &\cdot \prod_{i=1}^{m} p(k_l^i|L^i, ..., L^{i+s}, H, k_l^{i+1}, ..., k_l^{i+s}, k_h, lhs_j) \\ &\cdot \prod_{i=1}^{n} p(k_r^i|H, R^{i-s}, ..., R^i, k_h, k_r^{i-s}, ..., k_r^{i-1}, lhs_j) \end{aligned} \tag{7}$$

Different values of $s$ are then tried to determine how much horizontal context is helpful for the sentence compression task.

### 3.2.2 Lexicalization

The individual words included in a subtree can obviously contribute to whether or not that subtree should be dropped. Under the formulation so far, however, only the root of the subtree affects whether or not it should be dropped. *Lexicalization* is another technique from parsing that Galley and McKeown adopt for the sentence compression task. In a *lexicalized grammar*, each nonterminal is annotated with its head word and the part of speech tag of its head word. This therefore drastically increases the effective number of non-terminals.

Assume we are currently estimating the probability of dropping the prepositional phrase *with sauce* from our sentence. The effect of completely lexicalizing every non-terminal is shown in figure 2.

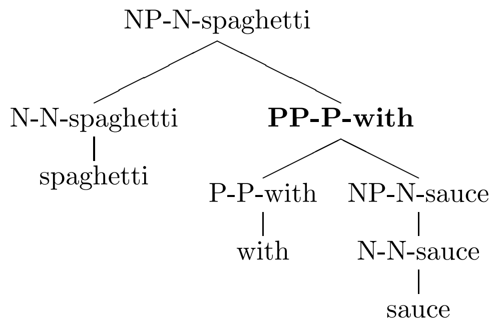


Figure 2: Lexicalization may help determine whether or not the bolded PP can safely be deleted.

The PP is a modifier to the right of the head, so without lexicalization, the probability of keeping this PP (according to equation 7) is:

$$p(k_r^1|k_h = 1, lhs_j = NP, H = N, R^1 = PP)$$

Under this formulation, dropping all prepositional phrases to the right of a noun are given the same probability. This is undesirable, as some prepositional phrases may be obligatory complements (in *the destruction of Rome*, *of Rome* is important information), while some prepositional phrases are optional (in *the scientist with red hair*, *with red hair* is not as crucial).

Galley and McKeown experiment with lexicalization by expanding the range of values that the variables $H$ and $L^i/R^i$ can take. Without lexicalization, $H$ takes the value of the syntactic category of the head of the rule. In the case of figure 2 and potentially deleting $PP$, $H = N$.

They find the head word of the subtree rooted at $H$ (*spaghetti*). Lexicalization is then incorporating by augmenting $H$ with either:

1. The part of speech category of this head word($H =$ *N-N*),
2. The head word ($H =$ *N-spaghetti*), or
3. Both the part of speech category of the head word and the head word itself ($H =$ *N-N-spaghetti*)

Besides expanding $H$, they also experiment with lexicalizing the constituent potentially to be dropped in the same way. For the same example of potentially dropping the prepositional phrase, without lexicalization, $R^1 = PP$. Including various levels of lexicalization, $R^1$ may be:

1. PP-P
2. PP-with
3. PP-P-with

With the maximum amount of lexicalization, our estimate for the probability of dropping the PP in figure 2 is now:

$$p(k_r^1|k_h = 1, lhs_j = NP, H = \textit{N-N-spaghetti}, R^1 = \textit{PP-P-with})$$

Note that the parent and other siblings besides the head are left unlexicalized.

### 3.2.3 Parent Annotation

With horizontal Markovization (Section 3.2.1), whether or not a node is dropped depends on some number of its siblings, and we can manipulate the number of siblings to condition on. With a probabilistic synchronous CFG, in the vertical direction, a node's inclusion or exclusion depends solely on its parent. However in natural language, there are certainly examples where information about more distant ancestors is useful. For example, a prepositional phrase which modifies the subject of the sentence (and so its grandparent is the $S$ node) is probably more likely to be kept than a prepositional phrase which modifies an $NP$ buried deeply in the tree. In order to incorporate these dependencies while remaining context-free, a common solution is to annotate each non-terminal with the label of its parent (Johnson, 1998). This is equivalent to splitting each non-terminal category into different categories for each possible parent. Galley and McKeown (2007) experiment with different levels of parent annotation, following Klein and Manning (2003)'s parsing experiments. For $v = 0$, no vertical history is conditioned on, for $v = 1$, only a parent is conditioned on (normal case), for $v = 2$, the parent and grandparent are conditioned on, etc.

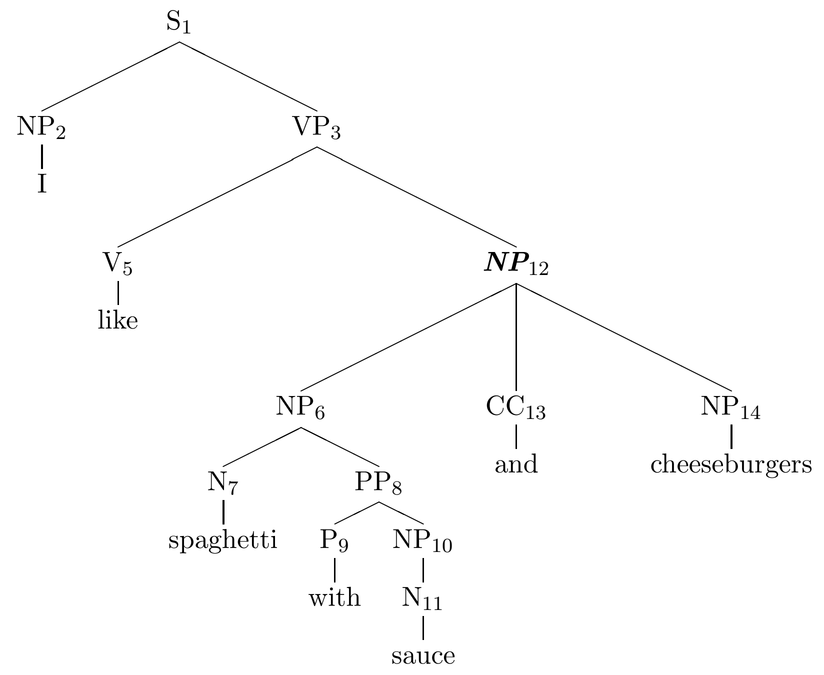
Returning to the case of dropping the $PP$ *with sauce*, the value of $lhs$ would vary between $\epsilon$, $NP$, NPˆVP, etc.

### 3.2.4 Better Rule Counts

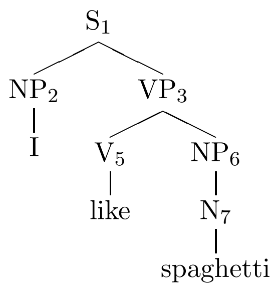
From a parallel corpus of aligned trees of long sentences and their compressed versions, we have mentioned before that one can simply read off the number of times each rule was used. However, there are two major limitations of this approach:

1. Very few sentences in abstracts are exact subsequences of a document sentence
2. Even if the words are subsequences of each other, the aligned trees might not be expressible with synchronous CFG rules

Rather than allowing unlimited deletions, 0 insertions, and 0 substitutions for finding training sentences, as in Knight and Marcu (2002), Galley and McKeown allow unlimited deletions, 0 insertions, but also up to 6 substitutions. In the Ziff-Davis corpus used by most work in sentence compression, only 1,087 (1.75%) of the sentences in abstracts can be used for sentence compression when restricted to have 0 substitutions. Their inclusion of sentences with substitutions expands the usable number of abstract sentences to 16,787 (25%).



(a) Long sentence parse



(b) Compressed version

Figure 3: It is not possible to align $NP_{12}$, and introducing *null* nodes will not help. This pair of trees can not derived with a synchronous CFG.

After extracting sentence pairs, both sentences in the pair are separately parsed with an automatic parser. Let $t_l$ be the parse tree of the long sentence and $t_c$ be the parse tree of the compressed sentence. The trees then need to be aligned to find the count of each synchronous CFG production rule. They use an approximation algorithm (Zhang and Shasha, 1989) to find the alignment between the nodes in the two trees that minimizes the edit distance between the two trees (in terms of insertions, substitutions, and deletions of nodes). After the alignments are done, they extract a rule for each linked non-terminal in the alignment, where the linked non-terminal is the left-hand side of the rule and the children in each tree make up the two right-hand sides.

These requirements will successfully extract rules from trees in which $t_c$ can be derived from $t_l$ solely by dropping constituents. However, there exist aligned trees in which the yield of $t_c$ is a subsequence of the yield of $t_l$, but the two trees cannot have been produced using synchronous CFG rules. An example of such a tree pair, which *cannot* be derived using synchronous CFG rules, is shown in Figure 3.

In derivations using synchronous CFGs, nodes in each tree are linked and derived together. The introduction of *null* nodes allows us to delete subtrees rooted at that node, however this means that in order to be expressible with a synchronous CFG, every node in $t_l$ must either have a corresponding node in $t_c$ or have the entire subtree rooted at that node deleted in $t_c$. Nodes which have descendants that are present in $t_c$ cannot be deleted. In figure 3, the presence of $NP_{12}$ in $t_l$ prevents the trees from being derived using synchronous CFG rules.

The example in figure 3 is actually a quite common case. Converting figure 3(b) into 3(a) cannot be done simply by adding constituents, but it could be done if we were allowed to "splice

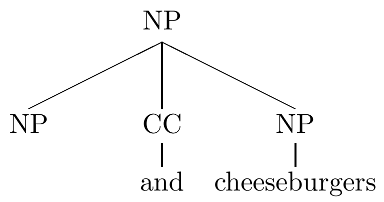


Figure 4: If this tree is adjoined into the tree in figure 3(b), then the trees in 3(a) and 3(b) could be aligned.

in" (i.e., use adjunction in Tree Adjoining Grammar (Joshi et al., 1975)) the tree in figure 4.

Tree Adjoining Grammar (TAG) adjunction operations, while not capable of being expressed in our synchronous CFG framework, often represent the insertion of optional material, which in sentence compression, is exactly what we want to delete.

Galley and McKeown (2007) are able to count some TAG adjunction cases by allowing a node to create a duplicate of itself as its parent inside the compressed tree. After adding this extra node to the compressed tree, the TAG adjunction operation can now be expressed with two synchronous CFG rules. For example, they would add an extra node $NP_{12}$ between $VP_3$ and $NP_6$ in figure 3(b). These trees would then increment the counts of $VP \rightarrow$ (*V NP, V NP*) and $NP \rightarrow$ (*NP CC NP, NP*).

### 3.2.5 Results

Galley and McKeown (2007) automatically evaluate the effect of different amounts of Markovization, lexicalization, and parent annotation, and then perform a manual evaluation with the best parameter settings to compare to prior work.

**Automatic Evaluation**: Overall, they found that according to their automatic measurement, lexicalization and increasing the amount of training data by allowing substitutions improved their performance, but Markovization and parent annotation did not. Automatic evaluation is done by computing word level accuracies of what is kept or deleted compared with the compressed sentences in the human-written abstracts.

Varying the amount of horizontal context, they find that conditioning on at least one sibling is better than conditioning on no siblings. The results for $s =$ 1,2, or 3 are all fairly similar. Increasing the amount of context to $s = 3$, which increases the sparsity of the rules, does not appear to hurt performance. Unfortunately, they do not compare against $s = \infty$ (no Markovization), so from their results, it is unclear whether Markovization was worthwhile to do at all for the sentence compression task.

The amount of parent annotation (vertical context) is varied from including 0 to 3 ancestors. They achieve their best accuracies with $v = 1$ (conditioning only on the parent). Unlike parsing, where parent annotation has been found to be helpful, for sentence compression, the best setting was when parent tags are left in their original form. This may partially be due to the amount of training data, as even Galley and McKeown (2007)'s largest training set contains only around 16,000 sentences. Possibly with more data, more annotation would become helpful.

Lexicalization and allowing substitutions in the paired training sentences are the two modifications that seem to have been beneficial. Increasing the number of substitutions allowed from 0 to 6 (and therefore multiplying the amount of training data by sixteen) increased word-level accuracies by about 2% for each level of lexicalization. The fact that their best performance was achieved with the maximum number of substitutions they tried implies that allowing even more substitutions would be worth trying.

Remember that the way lexicalization was done was by conditioning on subsets of {the category, the head POS, the head word} of the potential constituent to delete, as well as subsets of {the category, the head POS, the head word} of the head of that constituent's parent. Conditioning on all three (category, POS, and word) of the constituent to delete increases accuracy by about 3% over using the category alone. The lexicalization experiments were done under impoverished conditions though, with absolutely no vertical history and no horizontal history. A better experiment would have been to determine if lexicalization helped even when given the category of the parent and some of the siblings. Another issue is that some lexical information may be very domain-dependent. Within the same domain, the same words may show up in training and testing, but performance may be worse when tested on a new domain. While Knight and Marcu (2002) tested on both Ziff-Davis articles and articles from another source, Galley and McKeown (2007) only included the in-domain test set.

All of these results are only meaningful if one believes that the automatic metric is a good proxy for the task. Simple String Accuracy (which is almost equivalent to word-level accuracy) on the sentence compression task was found to be *not* significantly correlated with manual ratings of grammaticality and content (Clarke and Lapata, 2006b). If the word-level accuracy measure is not very correlated with performance on the sentence compression task, it unclear how meaningful the word-level accuracy results with different amounts of conditioning really are.

**Manual Evaluation**: According to Galley and McKeown, their best model is the one that conditions on the parent ($v = 1$), one sibling besides the head ($s = 1$), and all lexicalization (category, part of speech of the head word, and head word) for both the head and the constituent to be deleted. This model that they consider the best (partly by manually examining the output on a development set) has a lower word-level accuracy than several of the alternatives they tried. The compressed sentences produced by this best model are graded by manual raters as more grammatical, having better content, and are shorter than the sentences produced by the noisy channel model of Knight and Marcu (2002). The compression rate of Galley and McKeown (2007) is 63%, while the compression rate of the noisy channel model is 70%.

There are three main differences between Galley and McKeown (2007) and Knight and Marcu (2002):

- Estimating $p(c|l)$ directly instead of $p(c)p(l|c)$;
- Increasing the amount of training data by allowing substitutions in the compressed sentences;
- Using Markovization, parent annotation, and lexicalization to get different estimates of the probability of each rule.

While the paper focuses on the last contribution, it is unclear how much each of the three changes contributes to outperforming Knight and Marcu's noisy channel model. It would have been interesting if they had also compared against a model with: 0 substitutions in training data, no Markovization, only conditioning on the parent, and no lexicalization. The performance of this baseline model would have shown how much the performance benefited from estimating $p(c|l)$ directly, versus all of their other enhancements.

In the error analysis of Galley and McKeown (2007), they note that the worst compressions produced by their model are those in which the automatic parse of the long sentence contained several errors. One weakness of their approach is that since they only consider compressions that can come about through deleting constituents in the one-best parse of the long sentence, if that one-best parse is noisy, there is no way to recover. In the next section, we will discuss discriminative approaches that are designed to be more robust against noisy automatic parse trees.

## 4 Finding the Best Sentence: McDonald, 2006

### 4.1 Disadvantages of the Tree-based Approaches

The tree-based approaches of Knight and Marcu (2002) and Galley and McKeown (2007) first automatically parse the input sentence, and then create the compressed tree by deleting constituents in the tree. If the input sentence is parsed perfectly, this approach should often work. However, most sentences are ambiguous and have multiple possible parses. If the automatic parse is very wrong, then dropping "constituents" in the parse may lead to ungrammatical sentences.

Furthermore, the actual task finally requires a system to output a *sentence*, not a tree. The most probable tree does not necessarily yield the most probable sentence.

### 4.2 McDonald, 2006

The task is to produce a compressed sentence, so McDonald (2006) directly optimizes the quality of the compressed sentences produced, rather than being limited to making tree edits. The motivation is that the automatic parses produced may not be correct, and so the most probable tree (after deleting constituents from a bad parse) may not correspond to the most probable compressed sentence. McDonald still automatically parses the input, but then derives features from the parse tree, rather than using it to constrain the output. He also includes features from the words themselves, the part of speech tags, and a dependency parse, rather than using just the constituency parse tree. In order to combine these features, which are not independent, he uses discriminative learning to learn how to weight each of these features for the sentence compression task.

The number of possible compressions is exponential in the length of the sentence, so he defines features which factorize over words in the compressed sentence, so that dynamic programming can be used to find the top-scoring compression(s). Even though he is defining features over pairs of words, he manages to define syntactic features that provide information about when a constituent is being dropped (as in the tree-based approaches), yet are local to these pairs. We will discuss his decoding algorithm, the features used, the learning method, and then results.

#### 4.2.1 Decoding

In learning, we define a set of features $\Phi$ whose values depend on $l$ (the long sentence) and $c$ (the compressed sentence). The goal of discriminative learning is to learn a weight vector $w$, such that the maximum value of $w \cdot \Phi(l, c)$ occurs for the best $c$. Given that we have an exponential number of possible $c$'s for each $l$, we have two options if we want to avoid enumerating all compressions:

1. Define local features: Dynamic programming can exactly find the best scoring $c$

2. Define global features: Re-rank a partial list or use search; not guaranteed to find the best score

McDonald defines features that factorize over adjacent portions of compressed sentences, and so can use dynamic programming. Cohn and Lapata (2007) also choose to define local features (that decompose over derivation rules in the tree, as in Galley and McKeown (2007)) and so can also use dynamic programming to find the optimum. Riezler et al. (2003) and Nomoto (2008) instead use syntactic or dependency constraints to filter the set of compressions down to a manageable size and then use machine learning to choose the best out of that set; Nomoto (2009) uses a search over the possible compressions, exploring only a constant number at a time.

McDonald defines features over pairs of adjacent words in the *compressed* sentence. Leaving aside what the features are until section 4.2.2, we first describe how the optimal compression can be found using dynamic programming. The representation of the input and output are sequences of words, so $l = l_1, ..., l_n$ and $c = c_1, ..., c_m$[2]. He defines a function I such that I($c_i$) is the index of the corresponding word in $l$. All features are over the input and adjacent words in the output, so the score of a full compressed sentence $c$ is:

$$s(l, c) = \sum_{j=2}^{|c|} w \cdot \Phi(l, I(c_{j-1}), I(c_j))$$

Both $c$ and $l$ are defined to begin with -START- and end with -END-, so $c_1 = l_1 = -START-$ and $c_m = l_n = -END-$.

The maximum scoring compression can then be found using dynamic programming. He defines C[i] to be the maximum scoring sentence using a subsequence of the words $l_1, ..., l_i$ and including $l_i$. If $i = 1$, then the maximum scoring sentence including only the start symbol, i.e., an empty sentence, has a score of 0. Otherwise, if $i > 1$, then the best scoring sentence that includes $l_i$ must have one of $l_1, ..., l_{i-1}$ as the previous included word. The best score would then be the best of the score through that previous word plus the score of the new transition.

$$\begin{aligned} C[1] &= 0.0 \\ C[i] &= \max_{j<i} \; C[j] + w \cdot \Phi(l, j, i) \qquad \text{for i>1} \end{aligned} \tag{8}$$

Every sentence includes the ending symbol -END-, which is $C[n]$. $C[n]$ therefore contains the maximum score of a sentence including the ending symbol and including a subsequence of $l_1, ..., l_n$, so $C[n]$ contains the score of the best compression.

### 4.2.2 Features

McDonald includes *word*, *POS*, *constituency parse*, and *dependency parse* based features. Recall that all features must be defined over pairs of adjacent words in the compressed sentence.

All the features below are defined over $c_{j-1}$ and $c_j$, two adjacent words in the compression. Note that since $c_{j-1}$ and $c_j$ are adjacent and no reordering is allowed, any words $l_i$ such that

[2]McDonald refers to the long sentence as $x$ and the compressed sentence as $y$, but we continue to use $l$ and $c$ to maintain consistency

$I(c_{j-1}) < i < I(c_j)$ are words in the long sentence that have been dropped from the compression. We will refer to these intervening words as the *dropped words* for this pair.

**Word Features**: If any of the dropped words for this pair were verbs, then a feature indicating the identity of the verb is included. Additionally, there is a binary feature indicating whether or not any of the dropped words indicated negation. If the dropped words contain a mismatched parenthesis, another binary feature is set. A different binary feature is set if the dropped words completely form one parenthetical.

**Part of Speech (POS) Features**: There are features for the POS of $c_{j-1}$ and $c_j$ together, the POS of $c_{j-1}$, the POS of $c_j$, and the POS of $c_j$ and its neighbors in $l$. All of the above features are included both individually and conjoined with whether or not $c_{j-1}$ and $c_j$ were adjacent in $l$. He also includes features for the POS of each dropped word, both individually and conjoined with the POS of $c_{j-1}$ and $c_j$. Each dropped word also gets a feature for its POS and its neighbors in $l$.

**Constituency Parse Features**: For each node in the tree that a) dominates only dropped words, and b) its parent does not, he includes features for that node and its context (parent and adjacent siblings). For example, in the compressed sentence *I like spaghetti* and the original sentence *I really like spaghetti with sauce*, *spaghetti* and *-END-* would be adjacent in the compressed sentence. *with* and *sauce* are the two dropped words between the adjacent words. Given the syntactic tree in figure 1(a), the only node that is the highest ancestor that only dominates dropped leaves is the PP. Therefore he would record a feature that a PP would be dropped whose parent is NP when computing features for *spaghetti* and *-END-* being adjacent.

**Dependency Parse Features**: Dependency parses represent sentences in a different way than constituency parses. While in a constituency parse, the words are all at the leaves of the trees, with syntactic categories as internal nodes, in a dependency parse, every node (except the root) is a word. In a dependency parse tree, there is a directed edge from $a$ to $b$ if $b$ modifies $a$, and each word has exactly one parent. The dependency parse of our example sentence is shown in figure 5. For each dropped word, he includes a feature for the POS tag of its parent and a feature for the conjunction of its POS tag, its parent's POS tag, and whether the dropped word was a leaf. Let us again examine which features get set when *spaghetti* and *-END-* are adjacent, i.e., *with* and *sauce* are the dropped words. The parent of *with* is *spaghetti*, and *with* is a preposition (IN in the Penn Treebank tagset) *spaghetti* is a common noun (NN), and *with* is not a leaf in figure 5, so the following features are true: ParentPOS=NN and SelfPOS=IN_ParentPOS_=NN_WasLeaf=False. Dropping *sauce* sets the following: ParentPOS=IN and SelfPOS=NN_ParentPOS=IN_WasLeaf=True.

**Feature Analysis**: Given that McDonald defines features over bigrams in the output, it is very surprising he did not include the probability of these words being adjacent as a feature. He does not have any language model features. McDonald explains that he does not use any lexical features, because the training set is too small and so doing so would lead to overfitting. However, the more usual approach is *not* to estimate how likely two words are to be adjacent based on how often they were adjacent in the small labeled training set. Instead, one can gather large, possibly web-scale, amounts of *unlabeled* texts to measure how often those words are adjacent. McDonald cites a concern about overfitting as a rationale for not using lexical features. He is right to be concerned: if one includes the lexical items themselves as features, that is equivalent to adding $|Vocabulary|^2$ additional features. If instead a feature is defined as the count or probability of the two words according to some large background corpus, however, that is just one additional feature.

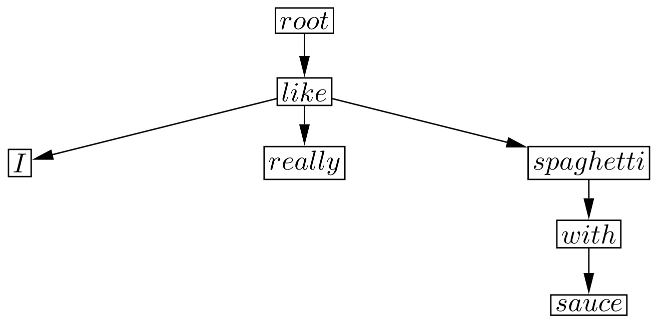


Figure 5: Dependency tree for *I really like spaghetti with sauce*

It is worth considering which aspects of Galley and McKeown (2007)'s model are captured by McDonald (2006)'s features and vice-versa. In the best model in Galley and McKeown (2007), whether or not to delete a constituent was conditioned on:

- The head that the constituent modifies. In the dependency tree, modifiers are children of the head that they modify. McDonald therefore has an approximation of this feature, by including the part of speech tag of its parent in the dependency tree.
- One modifier sibling towards the head. McDonald includes features for the left and right siblings in the constituency parse of the dropped constituent, so this feature is covered.
- Its parent. McDonald has a feature for the parent in the constituency parse of the dropped constituent, so this is also covered.
- Its head word. This is the first type of information that McDonald does not include. The only words which are features in McDonald's model are verbs, negation, and parentheses.
- The head word of the head that the constituent modifies. Again, McDonald does not include lexical features.

It is interesting that when we compare a tree-based model to a sentence-based model, all of the *syntactic* information has an approximation in McDonald (2006)'s model, while what is missing is the *lexical* information. A reason why Galley and McKeown (2007) were able to include lexical information is that they expanded the training set to allow substitutions. McDonald uses only the training set from Knight and Marcu (2002), around 1000 sentences, and so probably cannot compute any meaningful lexical statistics.

### 4.2.3 Learning

McDonald's scoring function for compressed sentences depends on a set of features and their weights. In the previous section, we discussed the set of features; in this section we will briefly explain how the weights are learned. McDonald uses the Margin Infused Relaxed Learning Algorithm (MIRA) (Crammer and Singer, 2003). MIRA is similar to perceptron learning in that it is an online learning method and it only updates weights on mistakes.

A feature of MIRA is that the learning update rate depends on a specified loss function $L$. After the update step, mistakes that are "far" from being correct need to have scores that are further away from the score of the correct answer, while mistakes that are "close" can have scores that are relatively closer to the score of the correct answer. McDonald is doing supervised learning, with the "correct answer" defined as the corresponding sentence in the human written abstract. His loss function for the compression task is the word-level error rate–the number of words either incorrectly kept or incorrectly dropped. As previously discussed, however, word-level accuracy is not correlated with compression quality (Section 3.2.5). If a loss function which was better correlated with manual ratings was used (for example, f-score over grammatical relations, introduced in section 5.4), he may have learned better weights.

After every example, MIRA solves a quadratic program, minimizing the size of the change to the weights, subject to one loss function-based constraint for each possible alternative compression $c'$. In the sentence compression task, if we created a constraint for every possible compression, we would have an exponential number of constraints. Therefore, he follows McDonald et al. (2005)'s set-up of MIRA for dependency parsing and only constructs constraints for the $k$-best (currently highest scoring) possible compressions.

### 4.2.4 Results

McDonald (2006) evaluates his compressions on the same set of data and using the same questions as Knight and Marcu (2002) and Galley and McKeown (2007). He compares against both the decision tree method in Knight and Marcu (2002) and the human compressions.

McDonald obtains better grammaticality and importance than the decision tree method. Because the ratings are done by different human judges, the average ratings of this work cannot be compared against the ratings reported in Galley and McKeown (2007). One issue to note is that because there are only 32 sentences in the test set, the differences between the different methods are probably not significant. This is an issue with any paper that follows the set-up of Knight and Marcu (2002).

As with Galley and McKeown (2007), overfitting might be an issue. While Knight and Marcu report similar results for their generative noisy channel model and their discriminative decision tree model on the 32 test sentences from the same source as the training set, when they test on a different genre of text, the noisy channel method does about the same, while the decision tree method experiences a substantial drop in both grammaticality and content of the produced sentences. It would therefore have been interesting if McDonald (2006) replicated this second experimental set-up of Knight and Marcu (2002) as well, to show how well his method generalized to new genres of text.

A major advantage of McDonald (2006)'s set-up over the tree-based generative models is that new features can easily be added to the model. If new features (for example, some of the discourse features discussed in section 5) were desired, they could simply be added to McDonald's list of features. The only restriction would be that they be formulated in such a way that they depend on only two adjacent words in the output.

The sentences produced by the model of McDonald (2006) actually receive higher importance ratings than the sentences from the abstracts (written by humans). He attributes this result to his model's producing longer sentences than the humans. This surprising finding underscores the importance of evaluating importance and grammaticality in tandem with compression rate. In the extreme case, a system could return sentences completely unedited, receiving perfect scores on grammaticality and importance but with no compression at all. Another aspect of evaluation to keep in mind is that the humans were producing these sentences as part of an abstract in context. Therefore the information the humans kept may be optimal for that context, but the evaluations are done using each sentence in isolation. We will return to this issue of context in the next section.

## 5 Discourse Approaches: Clarke and Lapata, 2007

The previously discussed papers all use the Ziff-Davis corpus compressed sentences in isolation (Knight and Marcu, 2002; Galley and McKeown, 2007; McDonald, 2006). Clarke and Lapata (2007) instead seek to compress sentences *in the context of the rest of the document.* Having the rest of the document available has an effect on how the *importance* of the words is measured. Facts which may appear to be very important when the sentence is presented in isolation may not be that crucial when the surrounding context is given, or vice-versa. One example of this is referring expressions (how entities are referred to). Consider the sentence: "U.S. President George W. Bush pushed out Treasury Secretary Paul ONeill and top economic adviser Lawrence Lindsey on Friday, launching the first shake-up of his administration to tackle the ailing economy before the 2004 election campaign." (example from Nenkova and McKeown (2003)). When this sentence is in isolation, the words "Treasury Secretary" provide important content and so should not be dropped. If this sentence is placed after a sentence which also mentions "Treasury Secretary Paul ONeill", then repeating the words "Treasury Secretary" again is unnecessary, and it would more natural to refer to him with only his last name (Nenkova and McKeown, 2003).

Besides keeping compressed sentences together with their content, another important difference between Clarke and Lapata (2007) and the previously discussed work is the training data used. While Galley and McKeown (2007) and McDonald (2006) both sought to learn how to compress sentences from pairs of compressed and uncompressed sentences, Clarke and Lapata (2007) do not use compressed sentence pairs at all. Instead, they do unsupervised learning. They formulate a measure of grammaticality and importance and seek to optimize it, subject also to constraints aimed at preserving grammaticality and constraints designed to keep important content. We first describe how they formulate the sentence compression task as an integer linear program (ILP), present the objective function they are trying to optimize, and then present the various constraints they came up with.

### 5.1 Sentence Compression as ILP

Similarly to McDonald (2006), Clarke and Lapata (2007) formulate the sentence compression task as making a binary decision for each word $l_i$ in the original sentence $l$ whether or not it should be included in the compressed sentence $c$. Clarke and Lapata convert it into an ILP problem by introducing several binary variables which indicate a) which word starts the compression, b) which words end the compression, c) which sequences of words are in the compression, and d) which words are in the compression :

$$\begin{aligned} p_i &= 1 \text{ iff } l_i \text{ starts the compression} \\ q_{ij} &= 1 \text{ iff the sequence } l_i\ l_j \text{ ends the compression} \\ x_{ijk} &= 1 \text{ iff the sequence } l_i\ l_j\ l_k \text{ is in the compression} \\ y_i &= 1 \text{ iff } l_i \text{ is in the compression} \end{aligned} \tag{9}$$

**Objective Function**: The objective function that they maximize is the sum of the probability of the compressed sentence according to a trigram language model and "significance scores" for the included words. The language model score rewards grammaticality, while the significance score rewards keeping content words which may be important.

A trigram language model is used in speech recognition, machine translation, and other applications to estimate the probability of a sequence of words, given a second-order Markov assumption, so that the probability of each word depends on the previous two. This probability can be used as a rough approximation of grammaticality, as sequences of words which are very improbable may be ungrammatical.

The log probability of a compressed sentence according to a trigram language model is:[3]

$$\begin{aligned} &\sum_{i=1}^{n} p_i \cdot \log p(l_i|start) \\ +&\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\sum_{k=j+1}^{n} x_{ijk} \cdot \log p(l_k|l_i,l_j) \\ +&\sum_{i=0}^{n-1}\sum_{j=i+1}^{n} q_{ij} \cdot \log p(end|l_i,l_j) \end{aligned} \tag{10}$$

The other half of the objective function is a significance score. The significance score is similar to tf-idf (term frequency-inverse document frequency, a weighting of words often used in information retrieval): words which are used often in the current document, but not very often in most documents, are considered more unique and therefore more important. For example, if a document mentions *dinosaurs* and *people* an equal number of times, the word *dinosaurs* is probably more important, because it is less frequent overall. Clarke and Lapata (2007) also weight words more highly if they are more embedded in the syntax of the sentence. This weighting may be appropriate with news texts, where there are often sentences of the form: "The company said that...", where the least embedded clause indicates attribution while the most embedded clause contains more important content. The significance score is:

$$\sum_{i=1}^{n} y_i \cdot \frac{d}{N} \cdot f_i \log \frac{F_a}{F_i}$$

[3]As presented in Clarke and Lapata (2007), it appears that they are maximizing the *sum* of the probabilities, which would be a surprising thing to do. In the authors' previous paper, Clarke and Lapata (2006a), they explicitly say that they are summing *log* probabilities, so that is what I assume they are doing here. Also, in the paper, in the triple summation, the index of $i$ begins at 1, not 0. I am assuming this is a typo, as if that were the case, the probability of a sequence of words would never include the probability of the second word, given the first word and the start symbol: i.e., p($w_1$,$w_2$,...,$w_n$) would be estimated by p($w_1$|start)p($w_3$|$w_1$,$w_2$)p($w_4$|$w_2$,$w_3$)...p(end|$w_{n-1}$,$w_n$).

Where $f_i$ is the frequency of $l_i$ in the current document, $F_i$ is the frequency of $l_i$ in a large background corpus, $F_a$ is the total number of content words in the corpus (and so constant for all words), and $\frac{d}{N}$ is the number of clauses $l_i$ is embedded in divided by the deepest level of embedding in the sentence. As defined earlier, $y_i$ is a binary variable indicating whether $l_i$ is included in the compression. The significance score is only defined over verbs (and sometimes nouns, discussed later) and is 0 for all other parts of speech.

The significance score in Clarke and Lapata (2007) comes from the significance score in Hori and Furui (2004), plus the syntactic weighting ($\frac{d}{N}$).

The overall objective function is therefore almost identical to that in Hori and Furui (2004). Hori and Furui (2004) compressed sentences from Japanese speech, by maximizing the sum of a trigram language model score, the significance scores of the included words, and the confidence of the automatic speech recognizer. Clarke and Lapata (2007) have almost the same objective function, except with a slightly different significance score and without the speech recognizer term. Hori and Furui (2004) used dynamic programming from left to right to identify the maximum scoring compressed sentence. If the goal was just to maximize this objective function, Clarke and Lapata (2007) could have used dynamic programming as well. However, Clarke and Lapata (2007) want to include some non-local constraints as well, so they formulate their problem as an ILP. We introduce the constraints in the next section.

## 5.2 Constraints

### 5.2.1 Non-discourse Constraints

Clarke and Lapata include a set of constraints to make sure the assignments to the variables defined in 9 are consistent with each other. For example, exactly one word should begin the sentence. Beyond these constraints, the rest of the constraints are aimed at ensuring grammaticality and content.

**Grammaticality Constraints**: The first constraint is that if a modifier is included, then what it modifies (its head) must also be included. This constraint is something that would often be learned by Galley and McKeown (2007), as they learn the probability of keeping any given modifier conditioned on whether or not its head was kept. However, Galley and McKeown must learn this separately for every part of speech. McDonald (2006) does not have any features that exactly correspond to not dropping modifiers. McDonald has features for the POS of modifiers and the POS of their heads (=parent in the dependency parse), but does not have any features for whether or not the head is dropped or included. Something else worth noting is that if a method was restricted to dropping subtrees in the *dependency* trees (along the lines of what Galley and McKeown do for constituency parse trees), then since each modifier's head is its parent, it could only be included if its head were included, and so this constraint would be satisfied automatically.

The second constraint for grammaticality enforces that a verb is present in the compression if and only if all of its arguments (subject and obligatory objects) are also present in the compression.

**Negation Constraint**: While McDonald included the dropping of negation as a feature, Clarke and Lapata include negation as a hard constraint, by enforcing that the word "not" is present in a compressed sentence if and only if the word it negated is also retained in the compressed sentence.

### 5.2.2 Discourse Constraints

Since Clarke and Lapata (2007) are not learning what words or constituents are important or unimportant from a parallel corpus of texts, they use several linguistically informed heuristics to determine which words are most important to keep. The next set of constraints reflect the intuition that the discourse context affects what information is most important in an individual sentence. While the discourse constraints should improve the coherence of the compressed document, the main motivation for including them is to better identify important content in the current sentence.

**Centering Theory Constraints**: Centering Theory (Grosz et al., 1995) is a model of local attentional state. The model claims that some entities in a sentence are more prominent than others, and that this difference in prominence affects whether or not entities can be referred to with pronouns, and also that which entities are shared in adjacent sentences affects local coherence.

Centering Theory defines *centers* as a set of entities which occur in a particular sentence and help to link that sentence to the rest of the discourse. Centers can therefore be either *forward-looking*, and potentially link this sentence with later sentences, or *backward-looking*, and link this sentence to the previous sentence. Centering Theory claims that for each utterance $U_i$ (defined here as a sentence) there is a single unique backwards center $C_b(U_i)$. The $C_b$ is defined as the entity which a) is mentioned in both the current sentence and previous sentence, and b) of these shared entities, was the one which was most prominent in the previous sentence. The forward looking centers $C_f(U_i)$ is a list of the entities in $U_i$ ordered by syntactic prominence.

Clarke and Lapata believe that Centering Theory can benefit sentence compression because keeping the entities which are shared across adjacent sentences will improve the coherence of the compressed document, while entities which are less prominent can be dropped. This definition of importance is therefore entirely determined by the surrounding sentences. If *I like spaghetti with sauce* is followed by a sentence about *spaghetti*, then *spaghetti* is marked as important to keep in this context; if instead the next sentence is about *sauce*, then the constraints will force *sauce* to be kept in the compression.

They automatically perform coreference resolution to determine which noun phrases refer to the same entities in the current sentence and the surrounding sentence. They also run a parser which outputs grammatical roles to rank the forward centers lists (subjects are ranked higher than direct objects are ranked higher than all other entities). They then automatically identify the backwards center for each sentence.

The backwards centers are what link the current sentence to the previous sentence. $C_b(U_i)$ therefore links $U_{i-1}$ and $U_i$. Note that by definition, $C_b(U_{i+1})$ is present in $U_i$, and so $C_b(U_{i+1})$ is what links $U_i$ and $U_{i+1}$. To keep the links to both the previous sentence and the following sentence, the Centering constraints are: include the *backwards center for the current sentence* and include the *backwards center of the next sentence.*

**Lexical Chains Constraints**: Coherence is also created through lexical cohesion and talking about the same thing (Halliday and Hasan, 1976). *Lexical chains* are sequences of related words which range over the entire document (Morris and Hirst, 1991). Morris and Hirst (1991) manually identified lexical chains in text, using a thesaurus to determine which words are related. Since then, automatic methods for identifying lexical chains have been proposed which use WordNet, an online lexical database (Barzilay and Elhadad, 1997; Silber and McCoy, 2002; Galley and McKeown, 2003).

This discourse constraint differs from the Centering Theory constraints in two main ways. First, lexical chains are a way of measuring the *global* discourse structure, while Centering Theory deals with *local* coherence. If, for example, other types of pasta are talked about much later in the document, those mentions may still contribute to *spaghetti* being marked as mandatory to retain. The second main difference is that related but distinct items (like *miles* and *feet*) can contribute to each other's importance, while the Centering Theory constraint requires references to the same entity for a noun phrase to be kept.

Clarke and Lapata automatically compute the lexical chains for the document currently being compressed. Chains are then ranked in decreasing order by the number of sentences they occur in. They then select the top half of these chains, which include words in the most sentences. They then add a new constraint that *any word in a top-scoring chain* must be retained in the compressed sentence.

**Pronoun Constraint**: This constraint is the most straightforward–*all personal pronouns* must be kept in the compression. Clarke and Lapata do not explain the motivation for this constraint. Since pronouns in text depend on the context for their interpretation, including pronouns increases the cohesion of a text (Halliday and Hasan, 1976).

**Back-off**: Note that all of the discourse constraints force words to be kept–none of the constraints force words to be deleted. They therefore apply the constraints in this order:

1. If $C_b(U_i)$ or $C_b(U_{i+1})$ exist, then only the Centering Theory discourse constraints apply.
2. Otherwise, if any words in top lexical chains are present in the sentence, then the lexical chains constraints apply.
3. Otherwise, if any pronouns are present, the pronoun constraint applies.
4. Otherwise, they include the significance scores for nouns, not just verbs, in the objective function being maximized.

## 5.3 Solving the ILP

They use the publicly available *lp_solve* program to solve their optimization function. Note that ILP is NP-hard, and they do *not* use an approximation function; instead they find the exact solution every time. As sentences generally contain at most tens of words, running time did not become an issue. In the journal version Clarke and Lapata (2008), they mention that while the coefficient constraint matrix is not totally unimodular, for every single one of their sentences, the optimal solution to their linear program had a completely integral solution.

## 5.4 Data and Evaluation

Clarke and Lapata (2007) decide not to use the Ziff-Davis corpus because their discourse constraints need the surrounding context. They create a new data set for sentence compression by giving annotators 82 full newspaper articles from the British National Corpus, the LA Times, and the Washington Post. The annotators were instructed to delete individual words from each sentence, while ensuring that they "(a) preserved the most important information in the source sentence, and (b) ensured the compressed sentence remained grammatical" (Clarke and Lapata, 2008).

While this data set has the nice property that sentences are not in isolation, the task used to elicit the compressions is quite artificial. The types of deletions made in an abstract are different from the types of deletions made in the context of the full article. Humans are more likely to drop long phrases in an abstract, while they usually restricted themselves to individual words when editing the news articles (Clarke and Lapata, 2008). As producing abstracts is a more likely final task than producing full length articles with missing words, Clarke and Lapata could have continued to use abstracts for their compressed sentences, but used the surrounding sentences in the *abstract* for their discourse constraints.

They evaluate their method both automatically and manually. Riezler et al. (2003) proposed using *grammatical relations* to automatically score compressions. The compressed sentence and original long sentence are both automatically parsed and (relation, head, modifier) triples automatically obtained. For example, from our original sentence, "(ncsubj like I)" would be extracted, indicating that "I" is a non-clausal subject and is the subject of the verb "like". A compressed sentence is then evaluated by its F-score over these relations. Clarke and Lapata (2006b) found that these scores are significantly correlated with human ratings.

The discourse ILP method scores higher according to this method than McDonald (2006). One fact that makes the comparison slightly unfair is that McDonald (2006) produced shorter compressions than Clarke and Lapata (2007). Both systems should be capable of producing compressions at a given compression rate, so they could have been evaluated at the same rate. McDonald explains how an additional variable for length can be added to his dynamic program to produce compressions with a pre-defined number of words. Clarke and Lapata could also easily adjust their compression rate by including one more constraint on the total number of included words.

The methods are also evaluated manually. Grammaticality is evaluated by human ratings, as is standard for this task. Rather than evaluating content with manual ratings though, they adapt an evaluation methodology from summarization. Annotators produce a set of questions which could be answered from the full articles. Human subjects are then given the automatically compressed articles and asked to answer these questions. The number of questions they are capable of answering correctly is then indicative of how much of the important information was retained in the compressed version. The compressions produced by Clarke and Lapata (2007) and McDonald (2006) are not significantly different on grammaticality ratings, but the humans given the discourse ILP compressions performed significantly better on the question answering task than the humans given McDonald (2006)'s compressions.

Given that the focus of Clarke and Lapata (2007) is on the inclusion of these discourse constraints, it is surprising they did not include an evaluation with and without the discourse constraints. With the results that they present, it is unclear whether the higher performance over McDonald (2006) is due to the discourse constraints, the formulation as an ILP problem, including language model probabilities, or simply having longer compressions. The rationale given for not presenting results with and without the discourse constraints is that the compression rates are too different. However, we have already mentioned how they could incorporate constraints to control the compression rate exactly.

### 5.4.1 Prior Work on Discourse and Sentence Compression

Clarke and Lapata (2007) incorporate discourse into their sentence-based sentence compression model. Daumé III and Marcu (2002) instead incorporate discourse into a tree-based model, using Rhetorical Structure Theory (Mann and Thompson, 1988), a very influential model of discourse.

Rhetorical Structure Theory (RST) is a theory of discourse which assumes that documents have a hierarchical structure and so can be represented as trees. The leaves of the tree are spans

| | Tree-based | Sentence-based |
|---|---|---|
| No discourse | Galley and McKeown (2007) | McDonald (2006) |
| With discourse | Daumé III and Marcu (2002) | Clarke and Lapata (2007) |

Table 1: A comparison of the approaches according to whether they find the best compressed tree or the best sentence and whether sentences are compressed in isolation or are affected by the surrounding discourse.

of text (often corresponding to clauses). Adjacent spans of text with a *rhetorical relations*, such as CONTRAST or ELABORATION, between them are recursively combined into a single node. The entire document is therefore represented as a single tree. The nodes in each rhetorical relation are labeled as either being the *nucleus* (more significant) or *satellite* (less significant). The intuition is that satellites can be dropped without much effect on the coherence of the rest of the text.

Daumé III and Marcu (2002) run an automatic discourse parser to create a single RST tree for the entire document. Each leaf of the tree is then parsed using a syntactic parser, creating a single tree incorporating both syntax and discourse. They adopt the noisy channel model (Knight and Marcu, 2002), designed for operations over sentence structures, to operate over both the discourse structure of a document and the structure of its individual sentences. The noisy channel model is able to drop constituents (Knight and Marcu, 2002), corresponding to subtrees in the syntactic tree. The discourse noisy channel model (Daumé III and Marcu, 2002) is able to drop subtrees in the mixed discourse/syntax tree. They compare the joint discourse and syntax noisy channel model to the syntactic noisy channel model. They find that grammaticality, coherence across sentences, and quality of the content are all improved when discourse is incorporated.

## 6 Discussion

We have focused on two main ways in which sentence compression approaches can vary:

- Representation: Some approaches find the best compressed syntactic tree, while others directly find the best compressed sentence.
- Discourse: Some approaches take only individual sentences as input, while others compress sentences in their discourse context.

A summary of the main papers discussed here and where they fall according to these two dimensions is presented in Table 1.

As mentioned in the introduction, the task of sentence compression requires a notion of what content is important and which compressed sentences are grammatical. In this section, we will compare Galley and McKeown (2007), McDonald (2006), and Clarke and Lapata (2007) and their representations of importance and grammaticality.

### 6.1 Importance

Galley and McKeown (2007) and McDonald (2006) learn what is important content to keep through supervised learning on pairs of compressed and uncompressed sentences, while Clarke and Lapata (2007) manually identify some hallmarks of importance and constrain the output to include these words.

Both Galley and McKeown (2007) and McDonald (2006) learn which syntactic categories are most important to keep and which syntactic categories are most often deleted during compressions. In contrast, Clarke and Lapata (2007) have no representation of which syntactic categories are most important.

On a lower level, there are probably some words which are important to keep whenever they occur. One example is negation–if a word like "not" is dropped, the meaning of the compressed sentence will change. Galley and McKeown (2007) have the only model which is capable of learning, for an arbitrary word, that it should be often be kept or deleted. However, given the small size of the training set, probably only a few "important words" would be learned. McDonald (2006) learns weights for negation and for individual verbs, so those are the only two types of words he can learn are important. Clarke and Lapata (2007), by contrast, force negation to be kept whenever what it modifies is kept, and (sometimes) force pronouns to be kept.

At a higher level, Clarke and Lapata (2007) use the rest of the document to identify important content, while Galley and McKeown (2007) and McDonald (2006) treat sentences in isolation and have no measure of global importance. In unsupervised content selection for summarization, the most common features used are frequency-based–the more often a term is repeated in a document, the more likely it is to be significant. Clarke and Lapata's "significance score" is a measure, similar to tf-idf, that gives a higher score for including words which are frequent in the current document but not very frequent in general. The other frequency-based feature of Clarke and Lapata is the lexical chains constraint–words which are synonyms or otherwise related to several other words throughout the document are kept. Finally, while the significance score and lexical chain constraint measure importance in the document globally, the Centering constraints force entities which are locally salient (based on the preceding and following sentence) to be kept.

## 6.2 Grammaticality

All grammatical sentences should have a well-formed syntactic parse tree. Galley and McKeown (2007) enforce this directly by only considering sentences which can be derived by dropping subtrees from the parse of the original sentence. McDonald (2006) and Clarke and Lapata (2007) do not create a parse tree for their compressed sentences.

Simply having a possible parse is not enough, however. For example, in English, all sentences are required to have a subject. While not explicitly required in any of the models, Galley and McKeown (2007) and McDonald (2006) may learn that NPs whose parent is S are very rarely dropped in the compressed versions. Clarke and Lapata (2007) do not have these kind of syntactic features, however they do have a constraint that if a verb is included, its subject must be as well.

Another way for a sentence to be ungrammatical is if what a verb subcategorizes for is not present. For example, the verb "give" should have a subject, a direct object, and an indirect object, so "I gave to Mary", which is missing a direct object, is ungrammatical. Different verbs subcategorize for different numbers and types of arguments. Therefore, McDonald (2006)'s approach, which did not include lexical information, would have no chance at capturing this aspect of grammaticality. Galley and McKeown (2007) include lexical information, but can only learn from the verbs which occur in the training set. If the verb "give" had not appeared in the Ziff-Davis sentence pairs, then they would not have learned that its indirect and direct object should both be kept. Clarke and Lapata (2007) have a hard constraint that if a verb is present in the compression then so are its subject and its objects. However, they do not mention if they distinguish between arguments (which are obligatory) and adjuncts (which are optional).

Speech recognizers, machine translation systems, and other NLP systems often use language models to approximate grammaticality. While language models can only detect local mistakes (all the trigrams in "What do you think Sarah will eat apples?" are fairly frequent), they can be a useful tool for detecting highly improbable sequences of words, which are likely to be ungrammatical. Only Clarke and Lapata (2007) use a language model to try to maximize grammaticality.

## 7 Conclusion and Future Work

We have presented an analysis of a variety of methods for sentence compression. Work in sentence compression can be categorized into approaches which find the most probable syntactic tree (Galley and McKeown, 2007) and those that find the best scoring sentence string directly (McDonald, 2006; Clarke and Lapata, 2007). Another distinction is that some approaches were supervised (Galley and McKeown, 2007; McDonald, 2006) and others were unsupervised (Clarke and Lapata, 2007).

One major limitation of the supervised approaches discussed here (Galley and McKeown, 2007; McDonald, 2006) is the very small amount of training data they use. However, there is no reason they should be limited to compressed and uncompressed pairs from Knight and Marcu (2002)'s set of abstracts. Pairs of articles and their human-written abstracts are abundant and can be gathered from scientific articles, the New York Times Annotated Corpus, and DUC/TREC summarization evaluations. Large amounts of training data are available without human annotators because writing abstracts, just like producing translations and transcriptions, is a "natural task routinely done every day for a real human need" (Halevy et al., 2009). These abstracts can be used to create sentence compression examples: Knight and Marcu (2002) show how abstract and article pairs can be automatically aligned, and Galley and McKeown (2007) show that relaxing this alignment by allowing substitutions leads to better performance.

Now that sentence compression has been shown to be feasible when sentences are compressed in isolation, future work may concentrate on incorporating sentence compression into summarization systems, in which conflicting discourse and content constraints may affect compressions. Lin (2003) showed that compressing the summary sentences individually with the noisy channel model of Knight and Marcu (2002) actually produced worse summaries than not compressing the sentences at all. There is hope for sentence compression improving summarization though; when an oracle is used to choose from among candidates of compressed versions, the compressed sentences do form a better summary than the uncompressed versions (Lin, 2003). Lin hypothesizes that sentence compression systems need to take the surrounding context into account in order to keep the important facts while removing global redundancy. While Clarke and Lapata (2007)'s compressions are influenced by the surrounding context, they are influenced by the surrounding context in the position the sentence was found *in the original article*. To apply sentence compression to summarization, one would want to compress sentences in the surrounding context *in the summary*. Additionally, rather than a pipelined approach of selecting sentences, ordering them, and then compressing them, models which jointly select sentences and compress them may yield better results for summarization (Daumé III and Marcu, 2002).

## 8 Acknowledgments

This report is a slightly revised version of my Written Preliminary Examination (WPE II). I would like to thank Mitch Marcus, Sudipto Guha, and Aravind Joshi for serving on my WPE II committee and for their helpful comments. I am also very grateful to Annie Louis, Arun Raghavan, and Michele Pitler for reading earlier drafts of this report.

## References

A.V. Aho and J.D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56.

R. Barzilay and M. Elhadad. 1997. Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization.*

D. Chiang. 2006. An introduction to synchronous grammars. Part of a tutorial given at ACL.

J. Clarke and M. Lapata. 2006a. Constraint-based sentence compression an integer programming approach. In *Proceedings of COLING/ACL*, pages 144–151.

J. Clarke and M. Lapata. 2006b. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *Proceedings of ACL*, pages 377–384.

J. Clarke and M. Lapata. 2007. Modelling compression with discourse constraints. In *Proceedings of EMNLP-CoNLL*, pages 1–11.

J. Clarke and M. Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31(1):399–429.

T. Cohn and M. Lapata. 2007. Large margin synchronous generation and its application to sentence compression. In *Proceedings of EMNLP/CoNLL*, pages 73–82.

T. Cohn and M. Lapata. 2008. Sentence compression beyond word deletion. In *Proceedings of COLING*, pages 137–144.

M. Collins. 1999. *Head-driven Statistical Models for Natural Language Parsing.* Ph.D. thesis, University of Pennsylvania.

S.H. Corston-Oliver and W.B. Dolan. 1999. Less is more: eliminating index terms from subordinate clauses. In *Proceedings of ACL*, pages 349–356.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

H. Daumé III and D. Marcu. 2002. A noisy-channel model for document compression. In *Proceedings of ACL*, pages 449–456.

M. Galley and K. McKeown. 2003. Improving word sense disambiguation in lexical chaining. In *Proceedings of IJCAI*, pages 1486–1488.

M. Galley and K. McKeown. 2007. Lexicalized markov grammars for sentence compression. In *Proceedings of NAACL/HLT*, pages 180–187.

G. Grefenstette. 1998. Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *Proceedings of the AAAI Spring Symposium on Intelligent Text Summarization*, pages 102–108.

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

A. Halevy, P. Norvig, and F. Pereira. 2009. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12.

M.A.K. Halliday and R. Hasan. 1976. *Cohesion in English.* Longman London.

C. Hori and S. Furui. 2004. Speech summarization: an approach through word extraction and a method for evaluation. *IEICE Transactions on Information and Systems*, 87:15–25.

H. Jing. 2000. Sentence reduction for automatic text summarization. In *Proceedings of the 6th Applied Natural Language Processing Conference*, pages 310–315.

M. Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

A.K. Joshi, L.S. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1):136–163.

D. Klein and C.D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, pages 423–430.

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

P. M. Lewis, II and R. E. Stearns. 1968. Syntax-directed transduction. *J. ACM*, 15(3):465–488.

C.Y. Lin. 2003. Improving summarization performance by sentence compression-a pilot study. In *Proceedings of the 6th International Workshop on Information Retrieval with Asian Languages*, pages 1–8.

N. Linke-Ellis. 1999. Closed captioning in America: Looking beyond compliance. In *Proceedings of the TAO Workshop on TV Closed Captions for the Hearing Impaired People, Tokyo, Japan*, pages 43–59.

W.C. Mann and S.A. Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281.

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98.

R. McDonald. 2006. Discriminative sentence compression with soft syntactic evidence. In *Proceedings of EACL*, pages 297–304.

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

A. Nenkova and K. McKeown. 2003. References to named entities: a corpus study. In *Proceedings of NAACL-HLT*, pages 70–72.

T. Nomoto. 2008. A generic sentence trimmer with CRFs. In *Proceedings of ACL*, pages 299–307.

T. Nomoto. 2009. A Comparison of Model Free versus Model Intensive Approaches to Sentence Compression. In *Proceedings of EMNLP*, pages 391–399.

S. Riezler, T.H. King, R. Crouch, and A. Zaenen. 2003. Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-functional grammar. In *Proceedings of NAACL*, pages 118–125.

J. Robert-Ribes, S. Pfeiffer, R. Ellison, and D. Burnham. 1999. Semi-automatic captioning of TV programs, an Australian perspective. In *Proceedings of TAO Workshop on TV Closed Captions for the Hearing-impaired People*, pages 87–100.

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

J. Turner and E. Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proceedings of ACL*, pages 290–297.

K. Zhang and D. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262.