Searching for Grammaticality: Propagating Dependencies in the Viterbi Algorithm

Stephen Wan[12] **Robert Dale**[1] **Mark Dras**[1]
[1]Centre for Language Technology
Div. of Information Communication Sciences
Macquarie University
Sydney, Australia
swan,rdale,madras@ics.mq.edu.au

Cécile Paris[2]
[2]Information and Communication Technologies
CSIRO
Sydney, Australia
Cecile.Paris@csiro.au

Abstract

In many text-to-text generation scenarios (for instance, summarisation), we encounter human-authored sentences that could be composed by recycling portions of related sentences to form new sentences. In this paper, we couch the generation of such sentences as a search problem. We investigate a statistical sentence generation method which recombines words to form new sentences. We propose an extension to the Viterbi algorithm designed to improve the grammaticality of generated sentences. Within a statistical framework, the extension favours those partially generated strings with a probable dependency tree structure. Our preliminary evaluations show that our approach generates less fragmented text than a bigram baseline.

1 Introduction

In abstract-like automatic summary generation, we often require the generation of new and previously unseen summary sentences given some closely related sentences from a source text. We refer to these as *Non-Verbatim Sentences*. These sentences are used instead of extracted sentences for a variety of reasons including improved conciseness and coherence.

The need for a mechanism to generate such sentences is supported by recent work showing that sentence extraction is not sufficient to account for the scope of written human summaries. Jing and McKeown [1999] found that only 42% of sentences from summaries of news text were extracted sentences. This is also supported by the work of Knight and Marcu [2002] (cited by [Daumé III and Marcu, 2004]), which finds that only 2.7% of human summary sentences are extracts. In our own work on United Nations Humanitarian Aid Proposals, we noticed that only 30% of sentences are extracted from the source document, either verbatim or with trivial string replacements.

While the figures do vary, it shows that additional mechanisms beyond sentence extraction are needed. In response to this, our general research problem is one in which given a set of related sentences, a single summary sentence is produced by recycling words from the input sentence set.

In this paper, we adopt a statistical technique to allow easy portability across domains. The Viterbi algorithm [Forney, 1973] is used to search for the best traversal of a network of words, effectively searching through the sea of possible word sequences. We modify the algorithm so that it narrows the search space not only to those sequences with a semblance of grammaticality (via *n*-grams), but further still to those that are grammatical sentences preserving the dependency structure found in the source material.

Consider the following ungrammatical word sequence typical of that produced by an *n*-gram generator, "*The quick brown fox jumped over the lazy dog slept by the log* ". One diagnosis of the problem is that the word *dog* is also used as the subject of the second verb *slept*. Ideally, we want to avoid such sequences since they introduce text fragments, in this case "*slept by the log*". We could, for example, record the fact that *dog* is already governed by the verb *jumped*, and thus avoid appending a second governing word *slept*.

To do so, our extension propagates dependency features during the search and uses these features to influence the choice of words suitable for appending to a partially generated sentence. Dependency relations are derived from shallow syntactic dependency structure [Kittredge and Mel'cuk, 1983]. Specifically, we use representations of relations between a head and modifier, ignoring relationship labels for the present.

Within the search process, we constrain the choice of future words by preferring words that are likely to attach to the dependency structure of the partially generated sentence. Thus, sequences with plausible structures are ranked higher.

The remainder of the paper is structured as follows. In Section 2, we describe the problem in detail and our approach. We outline our use of the Viterbi algorithm in Section 3. In Section 4, we describe how this is extended to cater for dependency features. We compare related research in Section 5. A preliminary evaluation is presented and discussed in Section 6. Finally, we conclude with future work in Section 7.

2 Narrowing the Search Space: A Description of the Statistical Sentence Generation Problem

In this work, sentence generation is essentially a search for the most probable sequence of words, given some source text.

However, this constitutes an enormous space which requires efficient searching. Whilst reducing a vocabulary to a suitable subset narrows this space somewhat, we can use statistical models, representing properties of language, to prune the search space of word sequences further to those strings that reflect real language usage. For example, *n*-gram models limit the word sequences examined to those that seem grammatically correct, at least for small windows of text.

However, *n*-grams alone often result in sentences that, whilst near-grammatical, are often just gibberish. When combined with a (word) content selection model, we narrow the search space even further to those sentences that appear to make sense. Accordingly, approaches such as Witbrock and Mittal [1999] and Wan et al. [2003] have investigated models that improve the choice of words in the sentence. Witbrock and Mittal's content model chooses words that make good headlines, whilst that of Wan et al. attempts to ensure that, given a short document like a news article, only words from sentences of the same subtopic are combined to form a new sentences. In this paper, we narrow the search space to those sequences that conserve dependency structures from within the input text.

Our algorithm extension essentially passes along the long-distance context of dependency head information of the preceding word sequence, in order to influence the choice of the next word appended to the sentence. This dependency structure is constructed statistically by an O(n) algorithm, which is folded into the Viterbi algorithm. Thus, the extension is in an O(n^4) algorithm. The use of dependency relations further constrains the search space. Competing paths through the search space are ranked taking into account the proposed dependency structures of the partially generated word sequences. Sentences with probable dependency structures are ranked higher. To model the probability of a dependency relation, we use the statistical dependency models inspired by those described in Collins [1996].

3 Using The Viterbi Algorithm for Sentence Generation

We assume that the reader is familiar with the Viterbi algorithm. The interested reader is referred to Manning and Schutze [1999] for a more complete description. Here, we summarise our re-implementation (described in [Wan *et al.*, 2003]) of the Viterbi algorithm for summary sentence generation, as first introduced by Witbrock and Mittal [1999].

In this work, we begin with a Hidden Markov Model (HMM) where the nodes (ie, states) of the graph are uniquely labelled with words from a relevant vocabulary. To obtain a suitable subset of the vocabulary, words are taken from a set of related sentences, such as those that might occur in a news article (as is the case for the original work by Witbrock and Mittal). In this work, we use the clusters of event related sentences from the Information Fusion work by Barzilay et al. [1999]. The edges between nodes in the HMM are typically weighted using bigram probabilities extracted from a related corpus.

The three probabilities of the unmodified Viterbi algorithm are defined as follows:

Transition Probability (using the Maximum Likelihood Estimate to model bigram probabilities)[1]:

$$p_{tr_{ngram}}(w_{i+1}|w_i) = \frac{count(w_i, w_{i+1})}{count(w_i)}$$

Emission Probability: (For the purposes of testing the new transition probability function described in Section 4, this is set to 1 in this paper):

$$p_{em}(w) = 1$$

Path Probability is defined recursively as:

$$p_{path}(w_0, \ldots, w_{i+1}) = p_{tr_{ngram}}(w_{i+1}|w_i) \times p_{em}(w) \times p_{path}(w_0 \ldots w_i)$$

The unmodified Viterbi algorithm as outlined here would generate word sequences just using a bigram model. As noted above, such sequences will often be ungrammatical.

4 A Mechanism for Propagating Dependency Features in the Extended Viterbi Algorithm

In our extension, we modify the definition of the Transition Probability such that not only do we consider bigram probabilities but also dependency-based transition probabilities. Examining the dependency head of the preceding string then allows us to consider long-distance context when appending a new word. The algorithm ranks highly those words with a plausible dependency relation to the preceding string, with respect to the source text being generated from (or summarised).

However, instead of considering just the likelihood of a dependency relation between adjacent pairs of words, we can consider the likelihood of a word attaching to the dependency tree structure of the partially generated sentence. Specifically, it is the rightmost root-to-leaf branch that can still be modified or governed by the appending of a new word to the string. This rightmost branch is stored as a stack. It is updated and propagated to the end of the path each time we add a word.

Thus, our extension has two components: *Dependency Transition* and *Head Stack Reduction*. Aside from these modifications, the Viterbi algorithm remains the same.

In the remaining subsections, we describe in detail how the dependency relations are computed and how the stack is reduced. In Figure 3, we present pseudo-code for the extended Viterbi algorithm.

4.1 Scoring a Dependency Transition

Dependency Parse Preprocessing of Source Text

The Dependency Transition is simply an additional weight on the HMM edge. The transition probability is the average of the two transition weights based on bigrams and dependencies:

$$p_{tr}(w_{i+1}|w_1) = average(p_{tr_{ngram}}(w_{i+1}|w_1), p_{tr_{dep}}(w_{i+1}|w_1))$$

Before we begin the generation process, we first use a dependency parser to parse all the sentences from the source text to

[1]Here the subscripts refer to the fact that this is a transition probability based on *n*-grams. We will later propose an alternative using dependency transitions.

obtain dependency trees. A traversal of each dependency tree yields all parent-child relationships, and we update an adjacency matrix of connectivity accordingly. Because the status of a word as a head or modifier depends on the word order in English, we consider relative word positions to determine if a relation has a forward or backward[2] direction. Forward and backward directional relations are stored in separate matrices. The Forward matrix stores relations in which the head is to the right of modifier in the sentence. Conversely, the Backward matrix stores relations in the head to left of the modifier. This distinction is required later in the stack reduction step.

As an example, given the two strings (using characters in lieu of words) "d b e a c" and "b e d c a" and the corresponding dependency trees:

we obtain the following adjacency matrices:

Forward (or Right-Direction) Adjacency Matrix

$$\begin{bmatrix} 0 & a & b & c & d & e \\ a & 0 & 1 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 1 & 0 \\ c & 0 & 0 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Backward (or Left-Direction) Adjacency Matrix

$$\begin{bmatrix} 0 & a & b & c & d & e \\ a & 0 & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 2 \\ c & 1 & 0 & 0 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We refer to the matrices as $Adj_{\mathrm{r}ight}$ and $Adj_{\mathrm{l}eft}$ respectively. The cell value in each matrix indicates the number of times word i (that is, the row index) governs word j (that is, the column index).

Computing the Dependency Transition Probability
We define the Dependency Transition weight as:

$$p_{tr_{dep}}(w_{i+1}|w_i) = p(Dep_{sym}(w_{i+1}, headStack(w_i))$$

where Dep_{sym} is the symmetric relation stating that some dependency relation occurs between a word and any of the words in the stack, irrespective of which is the head. Intuitively, the stack is a compressed representation of the dependency tree corresponding to the preceding words. The probability indicates how likely it is that the new word can attach itself to this incrementally built dependency tree, either as a modifier or a governer. Since the stack is cumulatively passed on at each point, we need only consider the stack stored at the preceding word.

This is estimated as follows:

$$p(Dep_{sym}(w_{i+1}, headStack(w_i))) = \max_{h \in headStack(w_i)} p(Dep_{sym}(w_{i+1}, h))$$

[2]These are defined analogously to similar concepts in Combinatory Categorial Grammar [Steedman, 2000].

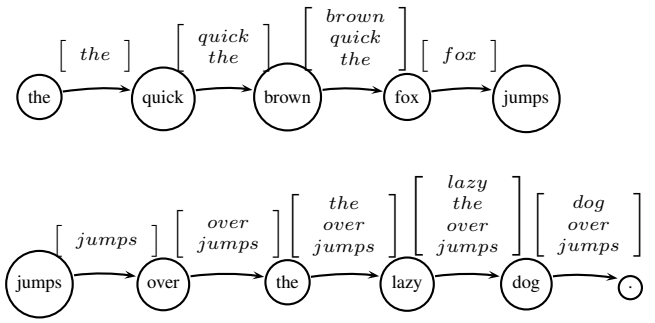

Figure 1: A path through a lattice. Although separated on two lines, it represents a single sequence of words. The stack (oriented upwards) grows and shrinks as we add words. Note that the modifiers to *dog* are popped off before it is pushed on. Note also that modifiers of existing items on the stack, such as *over* are merely pushed on. Words with no connection to previously seen stack items are also pushed on (eg. *quick*) in the hope that a head will be found later.

Here, we assume that a word can only attach to the tree once at a single node; hence, we find the node that maximises the probability of node attachment. The relationship $Dep_{sym}(a, b)$ is modelled using a simplified version of Collins' [1996] dependency model.

Because of the status of word as the head relies on the preservation of word order, we keep track of the directionality of a relation. For two words a and b where a precedes b in the generated string,

$$p(Dep_{sym}(a, b)) \approx \frac{Adj_{\mathrm{r}ight}(a, b) + Adj_{\mathrm{l}eft}(b, a)}{cnt(co\text{-}occur(a, b))}$$

where $Adj_{\mathrm{r}ight}$ and $Adj_{\mathrm{l}eft}$ are the right and left adjacency matrices. Recall that row indices are heads and column indices are modifiers.

4.2 Head Stack Reduction

Once we decide that a newly considered path is better than any other previously considered one, we update the head stack to represent the extended path. At any point in time, the stack represents the rightmost root-to-leaf branch of the dependency tree (for the generated sentence) that can still be modified or governed by concatenating new words to the string.[3] Within the stack, older words may be modified by newer words. Our rules for modifying the stack are designed to cater for a projective[4] dependency grammar.

There are three possible alternative outcomes of the reduction. The first is that the proposed top-of-stack (ToS) has no dependency relation to any of the existing stack items, in which case the stack remains unchanged. For the second and third cases, we check each item on the stack and keep a record

[3]Note that we can scan through the stack as well as push onto and pop from the top; this is thus the same type of stack as used in, for example, Nested Stack Automata.

[4]That is, if w_i depends on w_j, all words in between w_i and w_j are also dependent on w_j.

```
reduceHeadStack(aNode, aStack) returns aStack
Node_new ←aNode
Stack ←aStack # duplicate
Node_max ←NULL
Edge_prob ←0

# Find best chunk
While notEmpty(aStack)
  Head ←pop(aStack)
  if p(dep_sym (Node_new, Head)) > Edge_prob
    Node_max ←Head
    Edge_prob ←dep_sym(Node_new, Head)

# Keep only best chunk
While top(aStack) ≠ Node_max
  pop(aStack)

# Determine new head of existing string
if isReduced(Node_new, Node_max)
  pop(aStack)
else
  push(Node_new, aStack)
```

Figure 2: Pseudocode for the Head Stack Reduction operation

```
viterbiSearch(maxLength, stateGraph) returns bestPath
  numStates ←getNumStates(stateGraph)
  viterbi ←a matrix[numStates+2,maxLength+2]
  viterbi[0,0].score ←1.0
  for each time step t from 0 to maxLength do

    # Termination Condition
    if ((viterbi[endState, t].score ≠ 0)
      AND isAcceptable(endState.headStack))
        # Backtrace from endState and return path

    # Continue appending words
    for each state s from 0 to numStates do
      for each transition s' from s
        newScore ←
          viterbi[s,t].score × p_tr(s'|s) × p_em(s')
        if ((viterbi[s',t+1].score = 0) OR
            (newScore > viterbi[s', t+1]))
          viterbi[s',t+1].score ←newScore
          viterbi[s',t+1].headStack ←
            reduceHeadStack(s',viterbi[s,t].headStack)
          backPointer[s',t+1] ←s

  Backtrace from viterbi[endState,t] and return path
```

Figure 3: Extended Viterbi Algorithm

only of the best probable dependency between the proposed ToS and the appropriate stack item. The second outcome, then, is that the proposed ToS is the head of some item on the stack. All items up to and including that stack item are popped off and the proposed ToS is pushed on. The third outcome is that it modifies some item on the stack. All stack items up to (but not including) the stack item are popped off and the proposed ToS is pushed on. The pseudocode is presented in Figure 2. An example of stack manipulation is presented in Figure 1. We rely on two external functions. The first function, $dep_{sym}/2$, has already been presented above. The second function, $isReduced/2$, relies on an auxiliary function returning the probability of one word being governed by the other, given the relative order of the words. This is in essence our parsing step, determining which word governs the other. The function is defined as follows:

$$isReduced(w_1, w_2) = p(isHeadRight(w_1, w_2)) > p(isHeadLeft(w_1, w_2))$$

where w_1 precedes w_2, and:

$$p(isHeadRight(w_1, w_2)) \approx \frac{Adj_{\mathrm{right}}(w_1, w_2)}{cnt(hasRelation(w_1, w_2, \mathrm{where} i(w_1) < \mathrm{i}(w_2)))}$$

and similarly,

$$p(isHeadLeft(w_1, w_2)) \approx \frac{Adj_{\mathrm{left}}(w_2, w_1)}{cnt(hasRelation(w_1, w_2, \mathrm{where} i(w_1) < \mathrm{i}(w_2)))}$$

where $hasRelation/2$ is the number of times we see the two words in a dependency relation, and where $i(w_i)$ returns a word position in the corpus sentence. The function $isReduced/2$ makes calls to $p(isHeadRight/2)$ and$p(isHeadLeft/2)$. It returns true if the first parameter is the head of the second, and false otherwise. In the comparison, the denominator is constant. We thus need only the numerator in these auxiliary functions.

Collins' distance heuristics [1996] weight the probability of a dependency relation between two words based on the distance between them. We could implement a similar strategy by favouring small reductions in the head stack. Thus a reduction with a more recent stack item which is closer to the proposed ToS would be less penalised than an older one.

5 Related Work

There is a wealth of relevant research related to sentence generation. We focus here on a discussion of related work from statistical sentence generation and from summarisation.

In recent years, there has been a steady stream of research in statistical text generation. We focus here on work which generates sentences from some sentential semantic representation via a statistical method. For examples of related statistical sentence generators see Langkilde and Knight [1998] and Bangalore and Rambow [2000]. These approaches begin with a representation of sentence semantics that closely resembles that of a dependency tree. This semantic representation is turned into a word lattice. By ranking all traversals of this lattice using an *n*-gram model, the best surface realisation of the semantic representation is chosen. The system then searches for the best path through this lattice. Our approach differs in that we do not start with a semantic representation. Instead, we paraphrase the original text. We search for the best word sequence and dependency tree structure concurrently.

Research in summarisation has also addressed the problem of generating non-verbatim sentences; see [Jing and McKeown, 1999], [Barzilay *et al.*, 1999] and more recently [Daumé III and Marcu, 2004]. Jing presented a HMM for learning alignments between summary and source sentences trained using examples of summary sentences generated by humans. Daume III also provides a mechanism for subsentential alignment but allows for alignments between multiple sentences. Both these approaches provide models for later recombining sentence fragments. Our work differs primarily in granularity. Using words as a basic unit potentially offers greater flexibility in pseudo-paraphrase generation since we able to modify the word sequence within the phrase.

It should be noted, however, that a successful execution of our algorithm is likely to conserve constituent structure (ie. a coarser granularity) via the use of dependencies, whilst still making available a flexibility at the word level. Additionally, our use of dependencies allows us to generate not only a string but a dependency tree for that sentence.

6 Evaluation

In this section, we outline our preliminary evaluation of grammaticality in which we compare our dependency based generation method against a baseline. To study any improvements in grammaticality, we compare our dependency based generation method against a baseline consisting of sentences generated using bigram model.

In the evaluation, we do not use any smoothing algorithms for dependency counts. For both our approach and the baseline, Katz's Back-off smoothing algorithm is used for bigram probabilities.

For our evaluation cases, we use the Information Fusion data collected by [Barzilay *et al.*, 1999]. This data is made up of news articles that have been first grouped by topic, and then component sentences further clustered by similarity of events. There are 100 sentence clusters and on average there are 4 sentences per cluster. Each sentence in the cluster is initially passed through the Connexor dependency parser (www.connexor.com) to obtain dependency relations. Each sentence cluster forms an evaluation case in which we generate a single sentence. Example output and the original text of the cluster is presented in Figure 4.

To give both our approach and the baseline the greatest chance of generating a sentence, we obtain our bigrams from our evaluation cases.[5] Aside from this preprocessing to collect input sentence bigrams and dependencies, there is no training as such. For each evaluation case, both our system and the baseline method generates a set of answer strings, from 3 to 40 words in length.

For each generated output of a given sentence length, we count the number of times the Connexor parser resorts to returning partial parses. This count, albeit a noisy one, is used as our measure of ungrammaticality. We calculate the average ungrammaticality score across evaluation cases for each sentence length.

[5]Note that this is permissible in this case because we are not making any claims about the coverage of our model.

Original Text

A military transporter was scheduled to take off in the afternoon from Yokota air base on the outskirts of Tokyo and fly to Osaka with 37,000 blankets .

Mondale said the United States, which has been flying in blankets and is sending a team of quake relief experts, was prepared to do more if Japan requested .

United States forces based in Japan will take blankets to help earthquake survivors Thursday, in the U.S. military's first disaster relief operation in Japan since it set up bases here.

Our approach with Dependencies and End of Sentence Check

6: united states forces based in blankets

8: united states which has been flying in blankets

11: a military transporter was prepared to osaka with 37,000 blankets

18: mondale said the afternoon from yokota air base on the united states which has been flying in blankets

20: mondale said the outskirts of tokyo and is sending a military transporter was prepared to osaka with 37,000 blankets

23: united states forces based in the afternoon from yokota air base on the outskirts of tokyo and fly to osaka with 37,000 blankets

27: mondale said the afternoon from yokota air base on the outskirts of tokyo and is sending a military transporter was prepared to osaka with 37,000 blankets

29: united states which has been flying in the afternoon from yokota air base on the outskirts of tokyo and is sending a team of quake relief operation in blankets

31: united states which has been flying in the afternoon from yokota air base on the outskirts of tokyo and is sending a military transporter was prepared to osaka with 37,000 blankets

34: mondale said the afternoon from yokota air base on the united states which has been flying in the outskirts of tokyo and is sending a military transporter was prepared to osaka with 37,000 blankets

36: united states which has been flying in japan will take off in the afternoon from yokota air base on the outskirts of tokyo and is sending a military transporter was prepared to osaka with 37,000 blankets

Figure 4: A cluster of related sentences and sample generated output from our system. Leftmost numbers indicate sentence length.

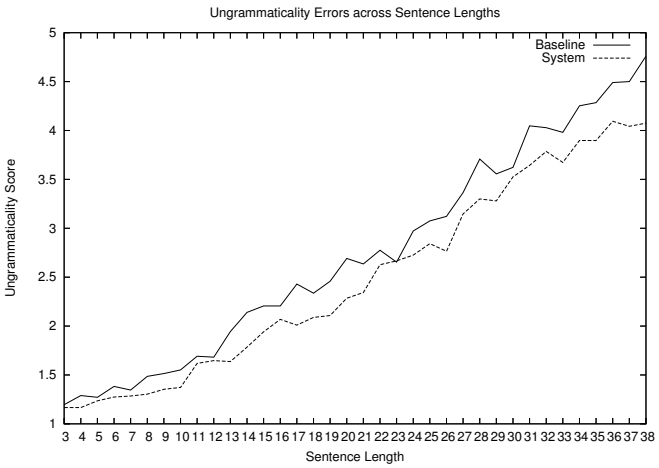

Figure 5: Ungrammaticality scores for generated output. Higher scores indicates worse performance.

The results are presented in Figure 5. Our approach almost always performs better than the baseline, producing less errors per sentence length. Using the Wilcoxon Signed Rank Text (alpha = 0.5), we found that for sentences of length greater than 12, the differences were usually significant.

7 Conclusion and Future Work

In this paper, we presented an extension to the Viterbi algorithm that statistically determines dependency structure of partially generated sentences and selects of words that are likely to attach to this structure. The resulting sentence is more grammatical than that generated using a bigram baseline. In future work, we intend to conduct experiments to see whether the smoothing approaches chosen are successful in parsing without introducing spurious dependency relations. We would also like to re-integrate the emission probability (that is, the word content selection model). We are also in the process of developing a measure of consistency. Finally, we intend to provide a comparison evaluation with Barzilay's Information Fusion work.

8 Acknowledgements

This work was funded by the Centre for Language Technology at Macquarie University and the CSIRO Information and Communication Technology Centre. We would like to thank the research groups of both organisations for useful comments and feedback.

References

[Bangalore and Rambow, 2000] Srinivas Bangalore and Owen Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th Conference on Computational Linguistics (COLING'2000), July 31 - August 4 2000*, Universität des Saarlandes, Saarbrücken, Germany, 2000.

[Barzilay *et al.*, 1999] Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th conference on Association for Computational Linguistics*, pages 550–557, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

[Collins, 1996] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, San Francisco, 1996. Morgan Kaufmann Publishers.

[Daumé III and Marcu, 2004] Hal Daumé III and Daniel Marcu. A phrase-based hmm approach to document/abstract alignment. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 119–126, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[Forney, 1973] G. David Forney. The viterbi algorithm. *Proceedings of The IEEE*, 61(3):268–278, 1973.

[Jing and McKeown, 1999] Hongyan Jing and Kathleen McKeown. The decomposition of human-written summary sentences. In *Research and Development in Information Retrieval*, pages 129–136, 1999.

[Kittredge and Mel'cuk, 1983] Richard I. Kittredge and Igor Mel'cuk. Towards a computable model of meaning-text relations within a natural sublanguage. In *IJCAI*, pages 657–659, 1983.

[Knight and Marcu, 2002] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artif. Intell.*, 139(1):91–107, 2002.

[Langkilde and Knight, 1998] Irene Langkilde and Kevin Knight. The practical value of N-grams in derivation. In Eduard Hovy, editor, *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 248–255, New Brunswick, New Jersey, 1998. Association for Computational Linguistics.

[Manning and Schütze, 1999] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[Steedman, 2000] Mark Steedman. *The syntactic process*. MIT Press, Cambridge, MA, USA, 2000.

[Wan *et al.*, 2003] Stephen Wan, Mark Dras, Cecile Paris, and Robert Dale. Using thematic information in statistical headline generation. In *The Proceedings of the Workshop on Multilingual Summarization and Question Answering at ACL 2003*, Sapporo, Japan, July 2003.

[Witbrock and Mittal, 1999] Michael J. Witbrock and Vibhu O. Mittal. Ultra-summarization (poster abstract): a statistical approach to generating highly condensed non-extractive summaries. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 315–316, New York, NY, USA, 1999. ACM Press.