Statistically Generated Summary Sentences: A Preliminary Evaluation using a Dependency Relation Precision Metric

Stephen Wan[1,2] **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

Often in summarisation, we are required to generate a summary sentence that incorporates the important elements of a related set of sentences. In this paper, we do this by using a statistical approach that combines models of *n*-grams and dependency structure. The approach is one in which words are recycled and re-combined to form a new sentence, one that is grammatical and that reflects the content of the source material. We use an extension to the Viterbi algorithm that generates a sequence that is not only the best *n*-gram word sequence, but also best replicates component dependency structures taken from the source text. In this paper, we describe the extension and outline a preliminary evaluation that measures dependency structure recall and precision in the generated string. We find that our approach achieves higher precision when compared to a bigram generator.

1 Introduction

The state-of-the-art in summarisation technology still rests heavily on the use of sentence extraction techniques. However, recent work has shown that sentence extraction is not sufficient to account for the scope of written human abstracts ([Jing and McKeown, 1999]; [Knight and Marcu, 2002]). In our own work on United Nations Humanitarian Aid Proposals, we noticed that only 30% of sentences from human authored abstracts could be extracted from the source document. Often non-extracted summary sentences are the result of recombining words and phrases from selected source document sentences. The end product is either a paraphrase or a fusion of information from component sentences.

Thus, to generate abstract-like summaries, we are motivated to explore a procedure that allows for a recombination of extracted words to form these new and previously unseen summary sentences, which we refer to as *Non-Verbatim Sentences*. Additionally, when we use such mechanisms for paraphrase generation and summarisation, we want to be assured that the generated summary accurately reflects the content of the source text from which it was generated.

For the purposes of easily porting the approach to various domains, we employ a statistical approach to text generation. We follow [Witbrock and Mittal, 1999] in using the Viterbi algorithm [Forney, 1973] to search through the sea of possible word sequences for summary sentences. We have developed an extension to this work that narrows the search space not only to those sequences that maintains a semblance of grammaticality (via *n*-grams), but further still to those that preserve dependency structure information found in the source material. This mechanism should result in improved grammaticality. Examples of generated sentences are shown in Figure 1.

As an example, consider the following ungrammatical word sequence typical of that produced by an bi-gram generator: *The relief workers distributed food shortages are an issue.* One explanation for the error is that the bigram *food shortages* occurred in a context where the word *food* is a modifier of the noun head *shortages*. However, an approach reliant on just bigrams would not be able to detect that the word *food* already has a governing head word, in this case *distributed*. Ideally, we want to avoid such sequences since they will result in text fragments, in this case *shortages are an issue*. We could, for example, record the fact that *food* is already governed by a verb and thus avoid appending a second governing word.

In addition to *n*-grams, the extension propagates dependency features to constrain the search space. A dependency representation of the partially generated sentence is used to influence the choice of the next word in the sentence. This representation is a simplified form of a shallow syntactic dependency structure [Kittredge and Mel'cuk, 1983] which has had dependency role labels discarded. To obtain this structure, a statistical (dependency-based) parsing mechanism is folded into the Viterbi algorithm, producing a dependency structure for each search path that is followed.

We think of dependency structure here as a poor but easily obtainable surrogate for a semantic representation. Though not ideal as a semantic representation (for example, it still contains auxiliary verbs and other surface syntactic features), this tree structure can be seen as encoding predicate-argument structure. Thus, as a by-product of the grammaticality mechanism in our extension, we hope that this will also improve the chances that the generated sentence reflects the content of the source text.

Ideally, we would determine if the generated sentence is entailed by the source sentences in order to gauge if it accurately reflects its content. However, the entailment mechanisms that might provide such an answer are always only as good as the knowledge bases available to perform the necessary inferences. Unfortunately, good knowledge bases are often hard to obtain.

Instead, a preliminary evaluation measures the recall and precision of dependency relations as a crude approximation to entailment. This evaluation says nothing about whether the generated sentence is true or not. For example, the omission of an adverbial relation (for example, negation) could have dire consequences on the truth value of the generated sentence.

Nevertheless, we can use this measurement to distinguish between grammatical but unrelated sentences and those that are related but whose truth value is unknown. That is, if this generated dependency structure replicates (if even partially) a reference dependency structure from a source sentence, we assume that this indicates (partial) evidence supporting the relatedness of the propositional content of the generated sentence. We call this set of sentences *Verisimilitudes*.

In the remainder of this paper, we discuss sentence generation as a search process in Section 2. Section 3 provides a detailed account of our extension to the Viterbi algorithm which considers a statistical model of dependency relations. In Section 4, we examine related work in the summarisation and paraphrase community. The evaluation and results are discussed in Section 5. Finally, we conclude with future work in Section 6.

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

In this work, sentence generation is couched as a search for the most probable sequence of words, given the vocabulary of 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 further prune the search space of word sequences 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.

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
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 1: A selection of example output. Sentences are prefixed by their length.

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 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 statistical dependency models inspired by those described in Collins [1996].

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

In this section, we present an overview of the main features of our algorithm extension. The Viterbi algorithm (for a comprehensive overview, see [Manning and Schütze, 1999]) is used to search for the best path across a network of nodes, where each node represents a word in the vocabulary. The best sentence is a string of words, each one emitted by the corresponding visited node on the path.

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). 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.

Arcs between nodes are weighted using two pieces of information: a bigram probability corresponding to that pair of words; and a probability corresponding to the likelihood of a dependency relation between that pair of words. Both probabilities are computed using Maximum Likelihood Estimation.

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 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, given the source text. To combine both the bigram and dependency models, we take the average of the dependency transition probability and the bigram probability.

To test only the effect of the dependency and *n*-gram-based transition probability in this evaluation, we assume that the emission probability is always one. The emission probability is interpreted as being a *Content Selection* mechanism that chooses words that are likely to be in a summary. Thus, in this paper, each word has an equally likely chance of being selected for the sentence.

Transition Probability is defined as:

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

where

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

Emission Probability (for this paper, always set to 1):

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

Path Probability is defined recursively as:

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

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

3.1 Preprocessing

When the system is given a new text to summarise, we first segment the text into sentences and then parse each sentence to obtain corresponding dependency parse trees. These trees are then used to populate two adjacency matrices. 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[1] direction.

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

The relief workers distributed food to the hungry.
The UN workers requested medicine and blankets.

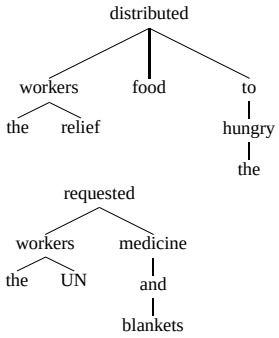

Figure 2: Two dependency trees from which the system will generate a sentence.

Forward and backward directional relations are stored in two separate matrices.

The first matrix, labelled Adj_{right}, stores (forward) relations in which the head of the relation is to the right of the modifier. The second matrix, Adj_{left}, stores (backward) relations in which the head is to the left of the modifier.

Presently, we only store dependency structures (both left and right) derived from the source text being summarised. One short-coming of this is that a dependency model built upon relationships in the source text will be very sparse. To allow the use of back-off mechanisms, we intend to keep track of a second set of adjacency matrices that would store dependency structures (again, both left and right) from a corpus. This is currently work in progress.

3.2 The Dependency Transition Probability

An Example Walkthrough

Given two input sentences *The relief workers distributed food to the hungry* and *The UN workers requested medicine and blankets*, and the corresponding dependency parses in Figure 2, the task is to generate a single sentence that contains material from these two sentences. As in [Barzilay *et al.*, 1999], we assume that the sentences stem from the same event, with the result that references can be fused together.

Imagine also that bigram frequencies have been collected from a relevant UN Humanitarian corpus. Figure 3 presents bigram probabilities and two sample paths through the lattice. The path could follow one of two forks after encountering the word *distributed*, since the corpus could have examples of the word pairs *distributed food* and *distributed blankets*. Since both *food* and *blankets* can reach the end-of-sentence state, both might conceivably be generated by considering just *n*-grams. However, only one is consistent with the input text.

To encourage the generation of verisimilitudes, we check for a dependency relation between *blankets* and *distributed* in the input sentence. As no evidence is found, we score this transition with a low weight. In contrast, there is evidence for the alternative path since the input text does contain a dependency relation between *food* and *distributed*.

Graph nodes:
w_1 is *workers*
w_2 is *distributed*
w_3 is *food*
w_4 is *blankets*
e is the *end-of-sentence* state

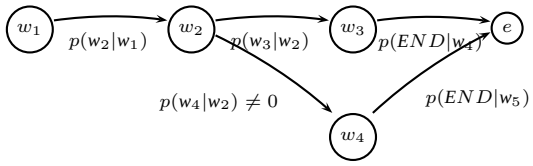

Figure 3: Two search paths. One is consistent with the input text, the other is not. Assume that the probabilities are taken from a relevant corpus such that $p(blankets|distributed)$ is not zero.

In reality, multiple words might still conceivably be modified by future words, not just the immediately preceding word. In this example, *distributed* is the root of a dependency tree structure representing the preceding string. However, any node along the rightmost root-to-leaf branch of the dependency tree (that represents the partially generated string) could be modified. This dependency structure is determined statistically using a probabilistic model of dependency relations. To represent the rightmost branch, we use a stack data structure (referred to as the *head stack*) whereby older stack items correspond to nodes closer to the root of the dependency tree.

Computing the Dependency Transition Probability
The probability of the dependency-based transition is estimated as follows:

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

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.

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 governing head of some node in the tree. Maximising this probability allows us to find the single best attachment point. Since the stack is cumulatively passed on at each point, we need only consider the stack stored at the preceding word.

The function $Dep_{sym}/2$ provides the probabilities of a dependency relation and accounts for the direction of the relation. For two words a and b where a precedes b in the generated string, we have

$$p(Dep_{sym}(a,b)) \approx \frac{Adj_{right}(a,b) + Adj_{left}(b,a)}{cnt(\text{co-occur}(a,b))}$$

where Adj_{right} and Adj_{left} are the right and left adjacency matrices taken from the source text. In these matrices, row indices are heads and column indices are modifiers. Items on the stack are annotated to indicate whether or not they already have a governing head. If the proposed dependency relation attempts to attach a new governing head for one which already has one, the probability is set to zero.

As mentioned above, to handle sparsity in the dependency model, we intend to use a back-off approach to smooth the dependency model. We envisage a linear combination of source text dependencies and corpus-based dependencies. We do note that incorporating any smoothing technique on a dependency model might improve grammaticality at the expense of conserving source text dependencies.

3.3 The Head Stack Representation

Maintaining Head Stack

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.[2] Within the stack, older words may be modified by newer words. Our rules for modifying the stack are designed to cater for a projective[3] 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 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 4. An example of stack manipulation is presented in Figure 5.

Although our extension caters for a right branching language such as English, we expect that an analogous algorithm exists for left branching languages, such as Japanese.

The algorithm in Figure 4 relies on three external functions. The first function, $dep_{sym}/2$, has already been presented above. The $annotateHeadedness/1$ function simply records if the input parameter is the head of the proposed dependency relation. The 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. In essence, this is our parsing step, determining which word governs the other. The function is defined as follows:

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

[2]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.

[3]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 attachment point
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)

# Remove subtree under attachment point
While top(aStack) ≠ Node_max
  pop(aStack)

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

annotateHeadedness(Node_new)
push(Node_new, aStack)
```

Figure 4: Pseudocode for the Head Stack Reduction operation

where w_1 precedes w_2, and:

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

and similarly,

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

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. In future work, we will use the same back-off procedure as described above, consulting the source material first and then the corpus.

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

An Example Walkthrough

We now step through the generation of the sentence *The UN relief workers distributed food to the hungry*. Figure 5 shows how the head stack update mechanism updates and propagates the stack of governing words as we append words to the path to produce this string.

We first append the determiner *the* to the new string and push it onto the empty stack. As dictated by a high *n*-gram probability, the word *UN* follows. However, there is no evidence of a relation with the preceding word, so we simply push it on the stack. Similarly, *relief* is appended and also pushed on the stack.

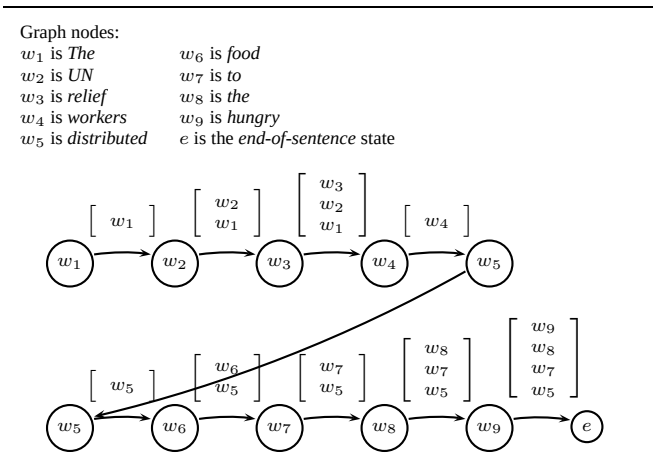

Figure 5: Propagating the head stack feature along the path.

When we encounter the word *workers* we find evidence that it governs each of the preceding three words. The modifiers are popped off and *workers* is pushed on. Skipping ahead, the transition *distribute food* has a high bigram probability and evidence for a dependency relation exists. This results in a strong overall path probability as opposed to the alternative fork in Figure 3. Since *distributed* can still be modified in the future by words, it is not popped off. The word *food* is pushed onto the stack as it too can still be modified.

The sentence could end there. Since we multiply path, transition and emission probabilities together, longer sentences will have a lower probability and will be penalised. However, we can choose to continue the generation process to produce a longer sentence. The word *to* modifies *distributed*. To prevent crossing dependencies, *food* is popped off the stack before pushing *to*. Appending the rest of the words is straightforward.

3.4 The Notion of Sentence-hood

Once a path reaches the *end-of-sentence* state, we can check the propagated head stack to see if it is a valid dependency structure.

Assuming that our *n*-gram model is accurate and the end-of-sentence marker is reached correctly, we can examine the head stack to see if the preceding string is acceptably structured. To do so, we simply perform a series of head stack reductions. Due to the way that we have defined the manipulation of the head stack so far, the only reductions possible here are cases where ToS is a modifier of an earlier stack object. Items are popped off the stack if this rule holds. After repeated reductions we should be left with a single word that represents the root of the dependency tree for the generated sentence. If this is the case, then the sentence is acceptable.

In practice, we set an upper bound on the sentence length, though the algorithm is free to terminate before then. The pseudocode to accept a sentence is presented in Figure 6. The extended Viterbi algorithm is presented in Figure 7.

```
isAcceptable(headStack) returns boolean
  while (headStack not empty)
    oldToS ←pop(headStack)
    if (not isHeadLeft(ToS,oldToS))
      return false
  if (isFinal(ToS))
    return true
  else
    return false
```

Figure 6: Pseudocode for accepting a generated sentence. The function $isFinal/1$ returns true if the word is suitable as the root of a dependency tree. For example, verbs and conjunctions would return true.

```
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 7: Extended Viterbi Algorithm

4 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 will briefly discuss 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.

Finally, instance-based generation mechanisms have been proposed by Varges [Varges, 2003] in which cosine similarity metrics are used to find the closest training instance to the current sentence being generated. The generation mechanism proceeds to minimise the distance from this training instance. Whereas Varges would use a grammar for generation, we produce our strings using *n*-grams.

5 Evaluation

In this section, we describe a preliminary experiment designed to evaluate whether a dependency-based statistical generator improves content overlap. We use a precision and recall styled metric on dependency relations. We find that our approach performs significantly better than the bigram baseline.

In the interests of minimising conflating factors in this comparison, we simplify the problem by building a bigram language model for each input cluster of text. This provides

both the bigram baseline and our system with the best possible chance of producing a grammatical sentence with the vocabulary of the input text. Note that the baseline is a difficult one to beat because it is likely to reproduce long sequences from the original sentences. However, an exact regurgitation of input sentences is not necessarily the outcome of the baseline generator since, for each cluster, the model is built from multiple sentences.

We do not use any smoothing algorithms for dependency counts in this evaluation since we do not back-off to corpora-based dependency probabilities at present time. Thus, given the sparseness arising from a small set of sentences, our dependency probabilities tend towards boolean values. For both our approach and the baseline, the bigrams are smoothed using Katz's back-off method.

The data for our evaluation cases is taken from the information fusion data collected by [Barzilay *et al.*, 1999]. This data is made up of news articles that have first been grouped by topic, and then component topic sentences further clustered by similarity of event. We use 100 sentence clusters and on average there are 4 sentences per cluster. Each sentence in a cluster is parsed using 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. For each evaluation case, the baseline method and our method generates a set of answer strings, from 1 to 40 words in length. The average sentence length is about 30 words. To minimise the number of confounding factors, we turned off the sentence-hood check in our system since it is the modified transition probability and head stack reduction mechanisms that we want to test.

The generated strings of the baseline and our system were then parsed by the Connexor parser. We compared the output dependency relations against those of the input cluster using a precision and recall metric.
The precision metric is as follows:

$$\text{precision} = \frac{\text{count(matched-relations)}}{\text{count(generated-relations)}}$$

The corresponding recall metric is defined as:

$$\text{recall} = \frac{\text{count(matched-relations)}}{\text{count(source-text-relations)}}$$

5.1 Results and Discussion

Figure 8 shows the average precision for each generated sentence length across all test cases. The results show that our dependency-based system scored better than a bigram baseline.

Using a two-tailed Wilcoxon test (alpha = 0.05), we found that the differences in precision scores are significant for most sentence lengths except lengths 17 and 32. The failure to reject the null hypothesis[4] for these lengths is interpreted as idiosyncratic in our data set. Ignoring the two outliers, we

[4]That is, the means of scores by our system and the baseline are not different.

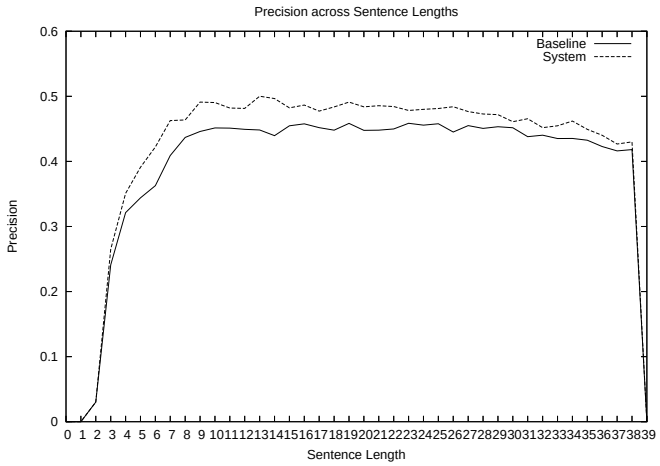

Figure 8: Dependency relation precision scores for generated output

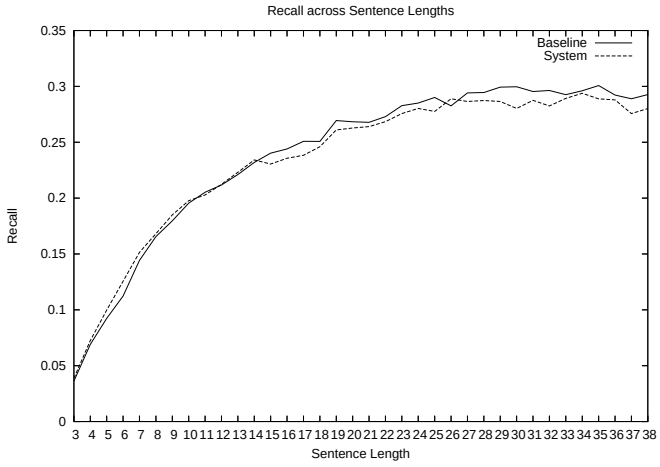

Figure 9: Dependency relation recall scores for generated output

reject the null hypothesis in general and conclude that using a dependency model in addition to a bigram model does improve precision.

The recall scores are shown in Figure 9. Unsurprisingly, there appears to be little difference between our approach and the baseline. Differences were not significant using the Wilcoxon test. This is consistent with the fact that our approach was designed to improve the content overlap and grammaticality of the generated sentence. As grammaticality was the key focus of the extension, there is no reason to believe that it should improve on the number of dependency relations recalled.

6 Conclusion and Future Work

In this paper, we have described our extension to the Viterbi algorithm that allows its use as a statistical generator that incorporates a dependency model. Sentences generated by this approach utilise dependency structures from the source text and are more grammatical than a bigram baseline. In addition, the mechanism also improves the degree of content overlap with the source text. In future work, we intend to explore the use of *n*-gram and dependency models built from a larger corpus in order to study their effects on grammaticality and content preservation. We also intend to compare our approach with the Information Fusion work of Barzilay [Barzilay *et al.*, 1999]. Finally, we simplified the content selection model in order to test the generation mechanisms. In future work, we intend to integrate previous content selection models with our dependency-based approach.

7 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.

[Varges, 2003] Sebastian Varges. *Instance-based Natural Language Generation*. PhD thesis, Institute for Communicating and Collaborative Systems, School of Informatics, University of Edinburgh, 2003.

[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.