Global Revision in Summarisation: Generating Novel Sentences with Prim's Algorithm

Stephen Wan[†‡] **Robert Dale**[†] **Mark Dras**[†]
[†]Centre for Language Technology
Div of Information Communication Sciences
Macquarie University
Sydney, NSW 2113
swan,madras,rdale@ics.mq.edu.au

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

Abstract

In abstract-like summarisation, extracted sentences containing key content are often revised to improve the coherence of the overall summary. In this work, we consider the task of Global Revision, in which a key sentence is revised and supplemented with additional content from the original document. Specifically, this task comprises two subtasks: selecting content; and grammatically ordering content, the focus of this paper. Using statistical dependency models, we search for a Maximal Spanning (Dependency) Tree that structures recycled words and phrases to form a novel sentence. Combining a modified version of Prim's algorithm with a four-gram language model, we evaluated our system on a sentence regeneration task obtaining Bleu scores of .30, a statistically significant improvement above the baseline.

1 Introduction

Faced with a seemingly endless supply of online textual information, it is often difficult to wade through a sea of documents returned by a search engine to identify those that are truly relevant to one's information need. To address this issue, automatic text summarisation technology strives to help provide shortened surrogates of documents that may be read quickly in order to make such judgements.

The current state-of-the-art approaches to summarisation simply return a list of extracted sentences. Whilst this is useful for identifying the gist of the document, this type of summary suffers from certain problems that make it difficult to glean more specific information about the original document. Typically, issues arise stemming from the fact that sentences are extracted in isolation and out of their original contexts. For example, an anaphoric reference in the extracted sentence may be missing the necessary antecedent to allow proper interpretation.

To combat such limitations, approaches to abstract-like summarisation often attempt to handle various additional linguistic phenomena in order to assist the reader make sense of the extracted pieces of information. Such methods range from accounting for discourse structure to maintain coherence (Marcu, 2000), to revising extracted text for fluency (for example, see Jing and McKeown (1999)). In this paper, we examine novel methods for accomplishing the task of revision as a text-to-text transformation.

Specifically, our overall goal is to perform what has been described as *Global Revision* (Cremmins, 1996). Global revision is the task of supplementing some key sentence, perhaps identified with a sentence extraction tool, with additional information from related sentences. These related sentences from the source document generally lie within the vicinity of the key sentence in the original document. Figure 1 presents such an example.

The summary sentence was written by a professional abstractor.[1] We hypothesise that it was composed by the abstractor on the basis of the three sentences shown below the summary, which we identified manually in the source document. In this example, the abstractor has paraphrased and conjoined the content of the first related sentence with the key sentence via the discourse marker *while*. Additionally, the noun phrase *conflict areas* replaces the phrase *inaccessible areas*. As can be seen by this example, global revisions tend to opportunistically recycle words and phrases from

[1]All text was taken from the United Nations website for Humanitarian Affairs at http://ochaonline3.un.org/humanitarianappeal/index.htm .

Summary Sentence:
There are presently an estimated 130,000 internally displaced persons (IDPs) in established camps, while it is estimated that 200,000 people are left behind in conflict areas with inadequate access to food or shelter.

Main Source Sentence:
There are presently an estimated 130,000 internally displaced persons (IDPs) in established camps.

Related Source Sentences:
The condition of an estimated 200,000 people left behind in inaccessible areas with inadequate access to food or shelter is much worse.

Conditions in conflict areas will worsen as the winter storms approach.

Figure 1: An example of global revision.

surrounding text where possible.

Broadly speaking, there are two primary tasks. The first is one of content selection in which, given a set of key and related sentences, the system must automatically identify which phrases and words are important enough to include in the final sentence. Often this is done by ranking words and phrases by some model of their salience. Models for content selection have been studied in previous work on text-to-text summarisation applications. For example, Witbrock and Mittal (1999) presented a model for learning how suitable words from a news article are for inclusion in headlines, or 'ultra-summaries'. In Wan et al. (2003), we presented models akin to that of latent semantic analysis for content selection in which words related to a single main theme were more likely to be selected for inclusion in a summary. Wan et al. (2005) presented methods aimed at recycling dependency relations from the original document, influencing which words are selected for inclusion in the generated summary.

In this paper, we hold the content selection factor constant and assume that one of the content models described above will be employed to rank words and phrases according to their importance. Here, we focus on the second part of the generation problem: namely, that of generating the final ordering of any highly ranked words and phrases. Essentially, the problem becomes one of choosing between the possible combinations of the selected strings, be they words or phrases.

Our baseline approach uses an *n*-gram language model to find the best ordering of words. At a local level, short generated sequences do indeed tend to look like valid word sequences. However, at a global sentence level, *n*-gram generation has no mechanisms to ensure that the sequence is well-formed enough to be grammatical. For example, there is no guarantee that the generated sequence will contain a verb, so it may not even correspond to a sentence.

To address this issue, we present a new method for combining the strengths of an *n*-gram language model (in this paper, we use four-gram models) with a statistical dependency model. Our hope is that grammaticality will be improved by including representations of global sentence structure during a search through different word orders. Our approach is to order an input set of strings (the recycled words and phrases) by attempting to construct a dependency tree that links these strings. This tree is built using a statistical model of dependency relations that is automatically obtained from the Penn Treebank (Marcus et al., 1994).

Our focus in this paper is to describe modifications to Prim's algorithm (Prim, 1957) for finding the maximal spanning tree, which we use to find the best dependency tree. An *n*-gram model is then used to find the best order of sibling nodes in the dependency tree when generating the string yield.

In the remainder of this paper, we begin by contrasting our approach to related work in Section 2. We discuss the Maximum Spanning Tree problem in more depth in Section 3, which presents our use of Prim's algorithm. In Section 4, we describe additional constraints to ensure that the generated tree is a dependency tree. Section 5 describes how to get the final word order once a spanning tree is found. Finally, before we conclude, our preliminary evaluations on the Wall Street Journal sections of the Penn Treebank are presented in Section 6.

2 Related Work

Because of the statistical nature of our work and the lack of symbolic representation as input, traditional text generation (for an overview, see Reiter and Dale 2000) is only peripherally related. Instead, we briefly review work in statistical text generation and paraphrase generation.

Statistical surface realisers (for example, Langkilde and Knight 1998; Bangalore and Rambow 2000) often start with a semantic representation for which a specific rendering, an ordering of words, must be determined. Such systems usually rank alternative word orderings using a language model to select the best sequence. Our approach blends content selection with surface realisation. That is, words and phrases will only be included in the final sentence if a suitable and natural ordering is possible. In our case, a model based on corpus statistics allows us to define what sounds natural.

Our ultimate goal of text-to-text summarisation is more similar to work on Information Fusion (Barzilay et al., 1999), a subproblem in multi-document summarisation. In this work, sentences presenting the same information are extracted from a set of news articles on the same news event, each perhaps published by a different news company. The basic premise of this work is that what is redundant across sentences is treated as important and worth reporting to the user. This in turn can be supplemented by additional non-repeated information if possible.

At first glance, at least in terms of input and output types (a set of sentences mapped to a single sentence), the Information Fusion and Global Revision tasks seem identical. In fact, it is the interrelationship of the sentences in the input set that differs between the two problems. Global Revision can be seen as the related single document version of Information Fusion. Crucially, the single document version does not deliver the same type of informational redundancy, and so we need different methods from those used in Information Fusion.

Other text-to-text approaches to generating novel sentences also aim to recycle sentence fragments where possible. More recently, work on phrase-based statistical machine translation has been applied to paraphrase generation (Bannard and Callison-Burch, 2005) and summarisation (Daumé III and Marcu, 2004). Whilst related, such methods benefit from the luxury of paired corpora in which one can learn automatically the transformation between the summary and the original document. What they do have in common with this work is a reliance on language modelling, though predominantly limited to *n*-gram models.

In this paper, we generate word sequences based on a dependency tree which we find using Maximal Spanning Tree algorithms. To our knowledge, this has not been done for text generation. However, the relationship between dependency parse trees and maximal spanning trees has been established in discriminative dependency parsing (McDonald et al., 2005). McDonald et al. specifically examine non-projective dependency trees, whereas this paper examines only projective ones. This difference is important in choosing a spanning tree algorithm.

Additionally, in parsing, one begins with a fixed sequence of words and the search is over tree structures. In contrast, we search for the best word sequence *and* tree structure concurrently, introducing issues that do not arise in parsing. These issues, concerning argument counts of words, are discussed in Section 4.

3 Understanding the Problem

Recall that our input is a set of words and phrases judged to be salient in content. Our task is to use as many as possible in a grammatical ordering corresponding to the novel summary sentence. To begin with, we preprocess the input strings such that each is represented by a single token. Phrases are treated as 'atomic' strings that cannot be broken up further. A phrase is represented by its syntactic head, an annotation that could be obtained automatically via an initial parse of each phrase. Drawing from the earlier example, the noun phrase *conflict areas* would thus be represented by *areas*. This converts the input from a set of strings of arbitrary length into one of single tokens.

Our approach is to find a dependency tree that connects the input head tokens. In searching for the best dependency tree, all possible dependency relations between pairs of words are considered. In effect, this collection of possible edges results in a maximally connected directional graph where each vertex is a word and edges represent a dependency relationship between the two words.

Edges are directional since it matters which word is the head and which is the child. Edges also have automatically calculated weights that represent the strength of the relation. Thus, an edge that is seemingly implausible as a dependency relation will have a low weight. In contrast, an edge representing a dependency relation that one observes frequently receives a high weight.

Finally, in this particular graph of possible dependency relations, we keep track of the relative ordering of the head and its child: either the head occurs to the left of the modifier in the word sequence, or to the right.

We base our calculations of dependency relation strength on the models proposed by Collins (1996). Our version of the models differs in that we do not account for distance between head and modifier words. We define a dependency relation event as having the following three attributes: the head token, the word token and its relative ordering, which can be *leftwards* or *rightwards*. Thus, the probability of the relationship for two words A and B in a direction D is defined as follows:

$$prob(A,B,D) = \frac{cnt(rel(A,B,D))}{cnt(co(A,B))}$$

where the frequency of seeing that relation is normalised by $co(A,B)$: the number of times we saw those words together in a sentence, i.e., a co-occurrence count.

We can back off to part-of-speech tags in two stages as Collins describes. The first stage is to alternate between using the part of speech for one word in the pairing:

$$prob_{1pos}(A,B,D) = \frac{cnt(rel(A_{pos},B,D)) + cnt(rel(A,B_{pos},D))}{cnt(co(A_{pos},B)) + cnt(co(A,B_{pos}))}$$

The final backoff stage is to use only parts-of-speech:

$$prob_{2pos}(A,B,D) = \frac{cnt(rel(A_{pos},B_{pos},D))}{cnt(co(A_{pos},B_{pos}))}$$

A combination of these backoff probabilities is done linearly:

$$\begin{aligned} &prob_{backoff1} = \\ &\quad \lambda_1 \times prob(A,B,D) + \\ &\quad (1-\lambda_1) \times prob_{backoff2}(A,B,D) \end{aligned}$$

where $prob_{backoff2}$ is in turn defined as:

$$\begin{aligned} &prob_{backoff2}(A,B,D) = \\ &\quad \lambda_2 \times prob_{1pos}(A,B,D) + \\ &\quad (1-\lambda_2) \times prob_{2pos}(A,B,Direction) \end{aligned}$$

where:

$$\lambda_1 = \frac{cnt(co(A,B))}{cnt(co(A,B)) + 1}$$

$$\lambda_2 = \frac{cnt(co(A_{pos},B)) + cnt(co(A,B_{pos}))}{cnt(co(A_{pos},B)) + cnt(co(A,B_{pos})) + 1}$$

```
01 add all vertices into RemainingNodes
02 BestEdges.store(root, $\langle$null,root$\rangle$)
03 loop while BestEdges not empty
04     currentEdge = $max_e$ (weight(e) : e $\exists$ BestEdges)
05     BestEdges.remove(currentEdge)
06     RemainingNodes.remove(modifier(currentEdge))
07     addEdgeToTree(currentEdge)
08     $\forall$ E=$\langle$ modifier(e), v $\rangle$ : v in RemainingNodes
09         $E_{best}$ = BestEdges.retrieve(v)
10         if weight($E_{best}$) $<$ weight(E)
11             BestEdges.store(modifier(E), E)
```

Figure 2: Prim's algorithm for Generating Dependency Trees

3.1 Using Prim's Algorithm

Ultimately, what we want is a dependency tree that contains all the nodes in the graph. However, there are numerous trees that might possibly provide such an appropriate structure. One method of defining an optimum tree is to score trees based on some function of the weights of its component edges and then rank them to choose the best one.

Finding a spanning tree that is maximal in the sum of its edge weights is a well studied problem in the general area of graph theory. One such solution is Prim's algorithm, which finds the exact solution by consecutively adding the best possible edge until all words are part of the tree. This should provide a projective dependency tree in which none of the dependency edges cross given the word sequence yield of the tree.

Our initial version of Prim's algorithm for finding optimal dependency trees is presented in Figure 2. We now step through the pseudo-code with a description outlining the intuitions for choosing this algorithm.

3.1.1 Intuitions Underlying the Algorithm

To begin with, a dummy root token is added to the spanning tree (line 2). The process starts by finding edges containing words that are likely to be the head of the main clause (line 8), namely those that have been observed in the corpus that have a relationship with the dummy root. These will be predominantly verbs. For these words, no alternative edges connecting them to the tree yet exist so we store them (line 11).

The best of the root-to-vertex edges is chosen on the next pass of the loop (line 4) and added to the spanning tree (line 7). From the newly appended word (likely to be verb), a larger subset of words is now reachable. In fact, due to the smoothed dependency model, all remaining words are considered as modifiers. That is, most relationships will have a non-zero weight. If this new edge beats any previously viewed edge to this word, we keep it and discard the worst one (line 10, 11). In this last step, we hope that words which previously had a tenuous attachment point to the tree, due to an improbable dependency relation, will now find a better head word to attach to.

Note that, when examining new edges in line 8, we consider dependencies in a leftward direction as separate from rightward relations. If both exist, then both are compared in line 10. In the end, only one alternative is kept depending on the strength of its weight.

The process loops until all nodes have been added to the spanning tree.

3.1.2 Hard Constraints

To complete the story, we used two additional heuristic filters which impose hard constraints on the tree structure. For simplicity, these are not listed in Figure 2. Because of our smoothing method, it is possible for words that are not verbs to end up attaching the root token. Since this is almost never the case, in this step we only considered words whose part-of-speech is 'verb' for this position.

The second constraint disallows a noun from modifying another noun. This rather brutal filter is an artifact of our experimental scenario in which nouns often represent full phrases. Adjoining a noun phrase to another can make the resulting string ungrammatical in this context.

4 Accounting for Argument Counts

Unfortunately, in the context of statistical generation, a maximal spanning tree isn't necessarily a well-formed dependency tree. The model of dependency relations and the search strategy based on Prim's algorithm treat dependency triples as independent events. However, this is not the case. One simple way in which the events are clearly not independent is that words have a minimum and maximum number of permissible modifying arguments as dictated by their grammatical category.

For example, prepositions generally take one and only one modifier in which the modifier points leftwards to the preposition. Similarly, a main clausal verb usually requires a subject as a minimum but could take on any number of additional arguments including, for example, a potentially endless list of adverbs. Resulting trees could thus have words missing arguments and words with too many arguments.

4.1 Argument Count Error Cases

We analysed the algorithm with several examples to see how such errors were being made. Three error cases were identified, which we have dubbed: Theft of a Good Argument; the Greediness Problem; and A Shortage of Words.

4.1.1 Theft of a Good Argument

In this particular case, the algorithm dictates that the newly attached node in the spanning tree should offer an attachment point to each of the remaining unattached nodes. Consider the case in Figure 3. The node $word_a$ might well be the only possible subject noun left for the verb. However, it may be the case that:

$$prob(word_b, word_a, D) > prob(verb, word_a, Left)$$

in which case, the node $word_b$ will steal away the much needed subject argument (line 10). In general, a word may be the last remaining possible argument filler for an obligatory argument slot of a parent word. As we have already considered edges leaving the original parent word at the time of its attachment to the tree, no further potential argument fillers will be examined. Thus, once stolen from, there may be no chance to fill obligatory argument slots.

4.1.2 The Greediness Problem

Figure 3 indicates that the node $word_b$ now has two nodes that could potentially attach to it. These may be the final attachments made to the spanning tree if they are the last two remaining nodes. However, if $word_b$ were in fact a word with limited argument slots, such as a preposition, it would now be over-saturated, which will ultimately result in an ill-formed dependency tree. In general, this may occur at any level of the tree if we greedily attach arguments to inner nodes.

4.1.3 A Shortage of Words

Conversely, some nodes may end up with insufficient arguments. This could happen at an inner node higher up in the tree if its arguments are stolen. It could also happen at the leaf level. If the last node to be attached to the spanning tree should have been an inner node, there will be no more words left to fill its argument slots. This might

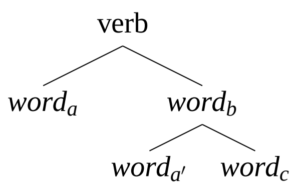

Figure 3: Argument error cases. $word_{a'}$ is 'stolen' node

happen if, for example, a preposition was the last word to be attached and it was not part of some phrasal verb.

4.2 A Model of Argument Counts

The model of argument counts and its usage in our modification of Prim's algorithm is designed to address the first two error cases. The third is left to future work since we cannot backtrack with this search algorithm. The model simply keeps track of the number of times a given word W has n modifiers, again accounting for leftward or rightward directions.

Given a word W, and an argument count n in the direction D, its probability is defined as:

$$prob_{arg}(W,n,D) = \frac{cnt_{arg}(W,n,D)}{cnt(W)}$$

We can compute a model that backs off to part-of-speech in an analogous way to the model defined for dependency relations.

$$\begin{aligned} &prob_{argbackoff}(W,n,D) = \\ &\quad \lambda \times (prob_{arg}(W,n,D)) \\ &\quad +(1-\lambda) \times (prob_{arg}(W_{pos},n,D)) \end{aligned}$$

Again, lambda is defined in terms of the denominator $cnt(W)$:

$$\lambda = \frac{cnt(W)}{cnt(W)+1}$$

4.3 Modifications to Prim's Algorithm

In our approach based on Prim's algorithm, attachment is final and thus only methods that attempt to prevent construction of an ill-formed tree are possible. That is, tree repair is not currently a part of our algorithm.

Given that Prim's algorithm does not allow backtracking, we have to re-weight edges using the model for argument counts during the search to avoid potential traps that would lead to an error case. Specifically, the argument count model is used in two places. Updating the priority queue to account for changing status in argument saturation attempts to combat the Greediness Problem. In choosing the best edge to keep, we can also avoid Thefts of a Good Argument.

Note that such a scheme would affect the edge weights dynamically as the search progressed. Since we rescore edge weights during search, they are obviously not fixed before the search begins, and thus Prim's algorithm is no longer guaranteed to produce the globally optimal solution.

4.3.1 Rescoring the Priority Queue

Recall that the BestEdges priority queue is checked at the beginning of the main loop of the algorithm to choose the next word to attach to the spanning tree. To avoid over-saturating a node, we adjust the 'attractiveness' of a dependency relation if its head word has peaked in terms of its optimum valency, possibly due to previous attachments.

To do so, we keep track of an argument count per word. When a modifier is added to the tree, the argument count for the head word is incremented by one. All remaining edges in the priority queue that attach to that head word must then be adjusted to account for this increment.

Re-weighting edges in BestEdge is computed as follows:

$$\begin{aligned} &prob_{adjust}(H,M,D) = \\ &\quad prob_{argbackoff}(H,n+1,D) \times prob(H,M,D) \end{aligned}$$

where n is the number of current children in direction D that the head H has in the spanning tree.

Typically, after the optimum number of edges for a word, additional edges are less probable or in fact ungrammatical, and the edge will correspondingly be adjusted to receive a lowered score. Once edges have been adjusted and the queue is sorted, the best edge is chosen and its modifier is attached to the spanning tree.

An additional advantage to re-weighting an edge in the priority queue is that it allows that edge to be more readily replaced with one that attaches the modifier to a head that still has available argument slots.

4.3.2 Choosing the Best Edge

When comparing alternative attachment points (line 8), we must choose between two competing parent nodes offering attachment points to the modifier in question. At the time of this comparison, argument thefts can also be avoided if we consider how many argument fillers the old parent still has. These are edges in the BestEdges priority queue that attach to the parent.

A new competing edge (with head H and modifier M) is re-weighted and given a lower weight if it would lead to a Theft of a Good Argument condition. The priority queue BestEdges stores P, the best attachment parent node seen so far for the modifier M. We must identify the potential for the valency of the old parent node, P, to be satisfied if M were stolen from it. Furthermore, the direction of its relationship, leftward and rightward, must also be accounted for such that there is a satisfaction score for each of these directions.

To do so, we find all edges in BestEdges with P as a head. These are divided into leftward and rightward sets, L and R. We first define the 'potential' for P to be satisfied. This is simply the arithmetic mean of the probabilities that W has 0 to m children, where m is either the size of L or R minus 1 respectively.

$$prob_{satisfied}(W,m,D) = \frac{\sum_{i=0}^{m} prob_{argbackoff}(W,i,D)}{m}$$

This allows us to re-weight the new competing edge (with head word H and modifier M in direction D) as:

$$prob_{re\text{-}weight}(H,M,D) = prob_{satisfied}(H,n,D) \times prob(H,M,D)$$

We then continue the comparison at line 10 with this re-weighted edge. Thus, if an old parent will not be satisfied after the modifier is stolen, then the new edge is penalised. If the parent node may still be satisfied, then the new edge is not disadvantaged.

5 Generating a Word Sequence

Once the tree has been generated, all that remains is to obtain the ordering of words based upon it. Because dependency relations in the tree are either of leftward or rightward direction, it becomes relatively trivial to order child nodes with respect to the parent. The only difficulty lies in finding a relative ordering for the leftward children, and similarly for the rightward children.

We propose two alternatives for finding the relative ordering of sibling nodes. The first is based on the strength of the dependency relation. This method sorts sibling nodes in descending order based on the final weight of the dependency relation as it was added to the tree. That is, the child node with the strongest dependency relation is placed closest to the parent node. For example, for leftward siblings, the relation with the highest edge comes just after (on the right of) the parent node. We will refer to this as a *Relation-based Ordering*.

The second method uses an n-gram language model to score word sequences between the strings yield of each sibling node. A greedy algorithm is used to determine the order of the phrases. For the set of rightward relations, the algorithm chooses the best word or phrase to follow a start-of-sentence marker based on n-gram probabilities. Once the best phrase is chosen to follow the start-of-sentence token, it then becomes the new point of comparison. Of the remaining phrases and words, we select the one that has the best n-gram probability following the most recently chosen phrase (or word). This continues until all input elements are consumed.

Where possible, an overlap window of size six $(2n-2)$ is examined, split at the point of the concatenation of the two phrases. That is, the last three words of the left phrase and the first three words of the right phrase. The probability of the concatenation is simply the probability of this substring which is in turn the product of the probabilities of four-grams within that substring. We will refer to this as a *N-Gram Ordering*.

Similarly, for the leftward relations, we use the greedy generation algorithm. However, instead of using the start-of-sentence marker, we begin with the phrase that realises the head node.

6 Evaluation

6.1 Task

In planning our evaluation, we sought to choose a task for which issues in content selection could be temporarily put on hold while we instead focused on appraising our approach for finding a grammatical ordering of input words and phrases. Thus, we simplified our task to that of ordering words and phrases from a single sentence.

As a surrogate measurement of grammaticality, we tested to see if we could regenerate the original string. In this task, manually delimited noun phrases and other words from a sentence are provided out-of-order as input to the generation module. The aim is to see how much of the original sentence can be regenerated. This can be measured by any string comparison method. In our case, we used the Bleu metric (Papineni et al., 2002).

This is actually an overly strict evaluation. A set of input words and phrases could have more than one grammatical ordering. If we were to generate one of the alternative orderings, our score would be adversely affected.

6.2 Training and Data

As we needed an annotated corpus of dependency relations, we used the Wall Street Journal portion of the Penn Treebank to obtain our dependency related frequencies. Dependencies were sourced from the events file of the Collins parser package which is in turn a compilation of the dependency events found in training sections 2-22 of the corpus. Development was done on section 00 and testing was performed on section 23.

A four-gram language model was also obtained from the same training data. In both baselines and in our system, we smooth the data using Katz's method which backs off to smaller sized *n*-grams.

As input, we used the tagged version of the WSJ sections which includes not only part-of-speech tags for each word but also delimitations on the base noun phrases, that is, noun phrases without nested constituents. We assumed that the last word of a base noun phrase is the head word of that phrase. Thus our input set of strings per sentence were the heads of base noun phrases and intervening single words.

6.3 Baselines

We decided to compare our system with two baselines. The first is the greedy *n*-gram ordering algorithm used to order rightward siblings in the dependency tree. Again, a start-of-sentence token is used and where possible a maximum overlap window of size six is used. As input, this baseline is given the same set of base noun phrases and words as our system. Noun phrases are not reduced to a single head word for this baseline generator. Thus, it too will receive the same benefits of word overlap due to a re-use of noun phrases. Note that this algorithm simply orders the input based on a language model and makes no use of tree structure.

When concatenating pairs of single tokens, the first baseline is disadvantaged because the maximum overlap window is limited to just a bigram.

System	Bleu
Tree	0.296
Tree+*N*-Grams	0.307
Tree+Arg	0.300
Greedy baseline	0.279
Viterbi baseline	0.107

Figure 4: Bleu scores

To account for this, we search with a full four-gram model over single tokens using a Viterbi-like algorithm with a beam of 100. As input, base noun phrases are also broken up into component words. Thus, this baseline tries to regenerate the test sentence based only on individual tokens, and does so without knowledge of noun phrase boundaries.

6.4 Results

We obtained favourable results, which indicate that an approach that orders words by first trying to build a dependency tree does produce more fluent text, as measured by Bleu. We present the Bleu scores in Table 4. The system named 'Tree' uses the simple version of Prim's algorithm presented in Figure 2 where sibling nodes are ordered based on the strength of their relations. The 'Tree+*N*-Gram' system differs only in that a language model and the greedy *n-gram* ordering algorithm are used to order siblings. The 'Tree+Arg' system uses the modifications for handling argument counts on top of the 'Tree+*N*-Gram' system.

Using the sign test and the sampling method as outlined in Collins et al. (2005), we tested for statistical significance. All differences are significant at $p < 0.01$ except between the variations of our systems. Although using *n*-grams to order sibling nodes obtained a slightly higher score, the lack of significance might be attributable to the fact that the dependency structure is responsible for most of the improvements in Bleu score and that the language model, when used in this manner, does not add much. Thus, in hindsight, it makes sense that we are less able to observe the effects of using the language model.

It also helps to examine the text that is actually generated by each system. In Figure 5, we present examples of typical generation cases for both the Tree+Arg system and the Greedy baseline.

Tree+Arg: a developer of mr. shidler 's company specializes in l.j. hooker and mr. simpson claims is in to have $ 1 billion a former senior executive commercial real-estate investment and assets

Greedy: in of is to have $ 1 billion in a former senior executive and commercial real-estate investment and claims assets a developer l.j. hooker specializes mr. simpson ; mr. shidler 's company

Original Sentence: mr. shidler 's company specializes in commercial real-estate investment and claims to have $1 billion in assets; mr. simpson is a developer and a former senior executive of l.j.hooker

Figure 5: Example generated sentences

6.5 Discussion and Future Work

Overall, we observed an improvement in the degree of string regeneration which we interpret as an improved ability on the part of our system to generate grammatical strings.

It is worth reflecting on the limitations of our approach both when the process operates as intended as well as when it performs poorly. By using corpus based dependency probabilities, we are biasing the system to recreate the semantic content of the corpus, insofar as our dependency representations are capable of capturing such information. Thus, in the evaluation, if the unseen test sentence which we want to regenerate should differ drastically in semantics from previously seen data, it will be difficult to recreate this sentence, even if the algorithm otherwise works as intended. For example, if a noun phrase that was always seen in the object position is suddenly in subject position in the test sentence, we will have little chance of regenerating the complete sentence. However, both our approach and our *n*-gram generation baselines would suffer from this.

Since we almost never recreate the original sentence completely, it is clear that our algorithm can still be improved. Considering the bigger picture, we have in effect shoe-horned Prim's algorithm into serving as an algorithm for finding an optimum dependency tree. As already discussed, dynamically weighting edges to account for satisfaction of argument counts results in the problem becoming more complex. Our modifications to handle argument counts seem still to be insufficient overall to avoid invalid dependency trees being considered.

Disappointingly, our modifications for handling argument counts seemed to have only a slight effect. One key limitation in the approach is the lack of a backtracking facility in the search algorithm. As noted above, at any given point in the search, what might have been the best edge to some node may suddenly become obsolete as a consequence of adding some other node to the tree. One option would be to keep an *n*-best list of edges instead of using a max function. We are currently working on such modifications to test for any improvements. It should be noted, however, that keeping an *n*-best list increases the complexity. We are also working to improve our model of argument counts further.

In our evaluation, noun phrases were conveniently delimited in the WSJ corpus. We intend to explore methods to either automatically delimit noun phrases or else to generate them from scratch. Finally, our evaluation in this paper focused only on string regeneration and used Bleu as a metric. We will also examine other metrics, such as the one proposed in Mutton et al. (2007), which evaluates the grammaticality of novel sentences. Additionally, we hope to evaluate our work in the context of the original task of global revision in abstract-like summarisation.

7 Conclusion

In this paper, we presented a mechanism for generating a novel sentence from a predetermined set of words. We argued that such a mechanism may be used for performing global revision as part of a larger text-to-text generation system for producing abstract-like summaries. Our approach employed a variation of Prim's algorithm to construct a dependency tree to be used in conjunction with a language model for ordering a set of words. The construction of the tree was performed using a statistical model of dependency relations. Our hope was that a language model would improve word order at the local level, whilst a syntactic model would enforce a valid grammatical structure at the global level. We observed a significant gain in performance over our baseline in a string regeneration task, attaining an system Bleu score of .307. We conclude that an approach that combines both syntactic and *n*-gram information can benefit from both types of information.

8 Acknowledgements

This work was funded by the Centre for Language Technology at Macquarie University and by the

CSIRO Information and Communication Technology Centre. We would like to thank both groups as well as the reviewers for useful comments and feedback.

References

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th Conference on Computational Linguistics*. Universität des Saarlandes, Saarbrücken, Germany.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, Michigan.

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

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, Morristown, NJ, USA.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. San Francisco.

Edward T. Cremmins. 1996. *The Art of Abstracting*. Information Resources Press.

Hal Daumé III and Daniel Marcu. 2004. A phrase-based hmm approach to document/abstract alignment. In *Proceedings of EMNLP 2004*, Barcelona, Spain.

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

Irene Langkilde and Kevin Knight. 1998. The practical value of N-grams in derivation. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, New Brunswick, New Jersey.

Daniel Marcu. 2000. *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Morristown, NJ, USA.

Andrew Mutton, Mark Dras, Stephen Wan, and Robert Dale. 2007. Gleu: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, USA.

R. C. Prim. 1957. Shortest connection networks and some generalisations. *Bell System Technical Journal*, 36:1389–1401.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge University Press, New York, NY, USA.

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

Stephen Wan, Robert Dale Mark Dras, and Ccile Paris. 2005. Towards statistical paraphrase generation: preliminary evaluations of grammaticality. In *Proceedings of The 3rd International Workshop on Paraphrasing*. Jeju Island, South Korea.

Michael J. Witbrock and Vibhu O. Mittal. 1999. Ultra-summarization: a statistical approach to generating highly condensed non-extractive summaries. In *Proceedings of the 22nd annual international ACM SIGIR conference*.