Improving Grammaticality in Statistical Sentence Generation: Introducing a Dependency Spanning Tree Algorithm with an Argument Satisfaction Model

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

Cécile Paris[‡]
[‡]ICT Centre
CSIRO
Sydney, Australia
Cecile.Paris@csiro.au

Abstract

Abstract-like text summarisation requires a means of producing novel summary sentences. In order to improve the grammaticality of the generated sentence, we model a global (sentence) level syntactic structure. We couch statistical sentence generation as a spanning tree problem in order to search for the best dependency tree spanning a set of chosen words. We also introduce a new search algorithm for this task that models argument satisfaction to improve the linguistic validity of the generated tree. We treat the allocation of modifiers to heads as a weighted bipartite graph matching (or assignment) problem, a well studied problem in graph theory. Using BLEU to measure performance on a string regeneration task, we found an improvement, illustrating the benefit of the spanning tree approach armed with an argument satisfaction model.

1 Introduction

Research in statistical novel sentence generation has the potential to extend the current capabilities of automatic text summarisation technology, moving from sentence extraction to abstract-like summarisation. In this paper, we describe a new algorithm that improves upon the grammaticality of statistically generated sentences, evaluated on a string regeneration task, which was first proposed as a surrogate for a grammaticality test by Bangalore et al. (2000). In this task, a system must regenerate the original sentence which has had its word order scrambled.

As an evaluation task, string regeneration reflects the issues that challenge the sentence generation components of machine translation, paraphrase generation, and summarisation systems (Soricut and Marcu, 2005). Our research in summarisation utilises the statistical generation algorithms described in this paper to generate novel summary sentences.

The goal of the string regeneration task is to recover a sentence once its words have been randomly ordered. Similarly, for a text-to-text generation scenario, the goal is to generate a sentence given an unordered list of words, typically using an *n*-gram language model to select the best word ordering. *N*-gram language models appear to do well at a *local* level when examining word sequences smaller than n. However, beyond this window size, the sequence is often ungrammatical. This is not surprising as these methods are unable to model grammaticality at the sentence level, unless the size of n is sufficiently large. In practice, the lack of sufficient training data means that n is often smaller than the average sentence length. Even if data exists, increasing the size of n corresponds to a higher degree polynomial complexity search for the best word sequence.

In response, we introduce an algorithm for searching for the best word sequence in a way that attempts to model grammaticality at the sentence level. Mirroring the use of spanning tree algorithms in parsing (McDonald et al., 2005), we present an approach to statistical sentence generation. Given a set of scrambled words, the approach searches for the most probable dependency tree, as defined by some corpus, such that it contains each word of the input set. The tree is then traversed to obtain the final word ordering.

In particular, we present two spanning tree algorithms. We first adapt the Chu-Liu-Edmonds (CLE) algorithm (see Chu and Liu (1965) and Edmonds (1967)), used in McDonald et al. (2005), to include a basic argument model, added to keep track of linear precedence between heads and modifiers. While our adapted version of the CLE algorithm finds an optimal spanning tree, this does

Proceedings of the 12th Conference of the European Chapter of the ACL, pages 852–860,
Athens, Greece, 30 March – 3 April 2009.

not always correspond with a linguistically valid dependency tree, primarily because it does not attempt to ensure that words in the tree have plausible numbers of arguments.

We propose an alternative dependency-spanning tree algorithm which uses a more fine-grained argument model representing argument positions. To find the best modifiers for argument positions, we treat the attachment of edges to the spanning tree as a weighted bipartite graph matching problem (or the *assignment* problem), a standard problem in graph theory.

The remainder of this paper is as follows. Section 2 outlines the graph representation of the spanning tree problem. We describe a standard spanning tree algorithm in Section 3. Section 4 defines a finer-grained argument model and presents a new dependency spanning tree search algorithm. We experiment to determine whether a global dependency structure, as found by our algorithm, improves performance on the string regeneration problem, presenting results in Section 5. Related work is presented in Section 6. Section 7 concludes that an argument model improves the linguistic plausibility of the generated trees, thus improving grammaticality in text generation.

2 A Graph Representation of Dependencies

In couching statistical generation as a spanning tree problem, this work is the generation analog of the parsing work by McDonald et al. (2005). Given a bag of words with no additional constraints, the aim is to produce a dependency tree containing the given words. Informally, as all dependency relations between each pair of words are possible, the set of all possible dependencies can be represented as a graph, as noted by McDonald et al. (2005). Our goal is to find the subset of these edges corresponding to a tree with maximum probability such that each vertex in the graph is visited once, thus including each word once. The resulting tree is a spanning tree, an acyclic graph which spans all vertices. The best tree is the one with an optimal overall score. We use negative log probabilities so that edge weights will correspond to costs. The overall score is the sum of the costs of the edges in the spanning tree, which we want to minimise. Hence, our problem is the minimum spanning tree (MST) problem.

We define a directed graph (digraph) in a standard way, $G = (V, E)$ where V is a set of vertices and $E \subseteq \{(u, v)|u, v \in V\}$ is a set of directed edges. For each sentence $w = w_1 \dots w_n$, we define the digraph $G_w = (V_w, E_w)$ where $V_w = \{w_0, w_1, \dots, w_n\}$, with w_0 a dummy root vertex, and $E_w = \{(u, v)|u \in V_w, v \in V_w \setminus \{w_0\}\}$.

The graph is fully connected (except for the root vertex w_0 which is only fully connected outwards) and is a representation of possible dependencies. For an edge (u, v), we refer to u as the head and v as the modifier.

We extend the original formulation of McDonald et al. (2005) by adding a notion of *argument positions* for a word, providing points to attach modifiers. Adopting an approach similar to Johnson (2007), we look at the direction (left or right) of the head with respect to the modifier; we consequently define a set $\mathcal{D} = \{l, r\}$ to represent this. Set $\mathcal{D}$ represents the linear precedence of the words in the dependency relation; consequently, it partially approximates the distinction between syntactic roles like *subject* and *object*.

Each edge has a pair of associated weights, one for each direction, defined by the function $s : E \times \mathcal{D} \rightarrow \mathbb{R}$, based on a probabilistic model of dependency relations. To calculate the edge weights, we adapt the definition of Collins (1996) to use direction rather than relation type (represented in the original as triples of non-terminals). Given a corpus, for some edge $e = (u, v) \in E$ and direction $d \in \mathcal{D}$, we calculate the edge weight as:

$$s((u, v), d) = -log\ prob_{dep}(u, v, d) \quad (1)$$

We define the set of part-of-speech (PoS) tags $\mathcal{P}$ and a function $pos : V \rightarrow \mathcal{P}$, which maps vertices (representing words) to their PoS, to calculate the probability of a dependency relation, defined as:

$$prob_{dep}(u, v, d) = \frac{cnt((u, pos(u)), (v, pos(v)), d)}{co\text{-}occurs((u, pos(u)), (v, pos(v)))} \quad (2)$$

where $cnt((u, pos(u)), (v, pos(v)), d)$ is the number of times where $(v, pos(v))$ and $(u, pos(u))$ are seen in a sentence in the training data, and $(v, pos(v))$ modifies $(u, pos(u))$ in direction d. The function $co\text{-}occurs((u, pos(u)), (v, pos(v)))$ returns the number of times that $(v, pos(v))$ and $(u, pos(u))$ are seen in a sentence in the training data. We adopt the same smoothing strategy as Collins (1996), which backs off to PoS for unseen dependency events.

3 Generation via Spanning Trees

3.1 The Chu-Liu Edmonds Algorithm

Given the graph $G_w = (V_w, E_w)$, the Chu-Liu Edmonds (CLE) algorithm finds a rooted directed spanning tree, specified by T_w, which is an acyclic set of edges in E_w minimising $\sum_{e \in T_w, d \in \mathcal{D}} s(e, d)$. The algorithm is presented as Algorithm 1.[1]

There are two stages to the algorithm. The first stage finds the best edge for each vertex, connecting it to another vertex. To do so, all *outgoing* edges of v, that is edges where v is a modifier, are considered, and the one with the best edge weight is chosen, where best is defined as the smallest cost. This minimisation step is used to ensure that each modifier has only one head.

If the chosen edges T_w produce a strongly connected subgraph $G_w^m = (V_w, T_w)$, then this is the MST. If not, a cycle amongst some subset of V_w must be handled in the second stage. Essentially, one edge in the cycle is removed to produce a subtree. This is done by finding the best edge to join some vertex in the cycle to the main tree. This has the effect of finding an alternative head for some word in the cycle. The edge to the original head is discarded (to maintain one head per modifier), turning the cycle into a subtree. When all cycles have been handled, applying a greedy edge selection once more will then yield the MST.

3.2 Generating a Word Sequence

Once the tree has been generated, all that remains is to obtain an 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 vertices with respect to a parent vertex. The only difficulty lies in finding a relative ordering for the leftward (to the parent) children, and similarly for the rightward (to the parent) children.

We traverse G_w^m using a greedy algorithm to order the siblings using an *n*-gram language model. Algorithm 2 describes the traversal in pseudo-code. The generated sentence is obtained by calling the algorithm with w_0 and T_w as parameters. The algorithm operates recursively if called on an inner node. If a vertex v is a leaf in the dependency tree, its string realisation *realise*(v) is returned.

[1] Adapted from (McDonald et al., 2005) and `http://www.ce.rit.edu/~sjyeec/dmst.html` . The difference concerns the direction of the edge and the edge weight function. We have also folded the function 'contract' in McDonald et al. (2005) into the main algorithm. Again following that work, we treat the function s as a data structure permitting storage of updated edge weights.

```
   /* initialisation */
1  Discard the edges exiting the w_0 if any.
   /* Chu-Liu/Edmonds Algorithm */
2  begin
3      T_w ← (u,v) ∈ E : ∀_{v∈V,d∈D} arg min_{(u,v)} s((u,v),d)
4      if M_w = (V_w, T_w) has no cycles then return M_w
5      forall C ⊂ T_w : C is a cycle in M_w do
6          (e,d) ← arg min_{e*,d*} s(e*,d*) : e ∈ C
7          forall c = (v_h, v_m, ) ∈ C and d_c ∈ D do
8              forall e' = (v_i, v_m) ∈ E and d' ∈ D do
9                  s(e',d') ← s(e',d') − s(c,d_c) − s(e,d)
10             end
11         end
12         s(e,d) ← s(e,d) + 1
13     end
14     T_w ← (u,v) ∈ E : ∀_{v∈V,d∈D} arg min_{(u,v)} s((u,v),d)
15     return M_w
16 end
```

Algorithm 1: The pseudo-code for the Chu-Liu Edmonds algorithm with our adaptation to include linear precedence.

We keep track of ordered siblings with two lists, one for each direction. If the sibling set is leftwards, the ordered list, R_l, is initialised to be the singleton set containing a dummy start token with an empty realisation. If the sibling set is rightwards then the ordered list, R_r is initialised to be the realisation of the parent.

For some sibling set $C \subseteq V_w$ to be ordered, the algorithm chooses the next vertex, $v \in C$, to insert into the appropriate ordered list, $R_x, x \in \mathcal{D}$, by maximising the probability of the string of words that would result if the realisation, *realise*(v), were concatenated with R_x.

The probability of the concatenation is calculated based on a window of words around the join. This window length is defined to be $2 \times \text{floor}(n/2)$, for some n, in this case, 4.

If the siblings are leftwards, the window consists of the last *min*$(n-1, |R_l|)$ previously chosen words concatenated with the first *min*$(n-1, |$*realise*$(v)|)$. If the siblings are rightwards, the window consists of the last *min*$(n-1, |$*realise*$(v)|)$ previously chosen words concatenated with the first *min*$(n-1, |R_r|)$. The probability of a window of words, $w_0 \ldots w_j$, of length $j+1$ is defined by the following equation:

$$\textit{prob}_{LMO}(w_0 \ldots w_j) = \prod_{i=0}^{j-k-1} \textit{prob}_{MLE}(w_{i+k} | w_i \ldots w_{i+k-1}) \quad (3)$$

```
/* LMO Algorithm */
input : v, T_w where v ∈ V_w
output: R ⊆ V_w
1  begin
2      if isLeaf(v) then
3          return {realise(v)}
4      end
5      else
6          C_l ← getLeftChildren(v, T_w)
7          C_r ← getRightChildren(v, T_w)
8          R_l ← {start}
9          R_r ← {realise(v)}
10         while C_l ≠ {} do
               c ← arg max_{c∈C_l} prob_ngram(LMO(c, T_w) ∪ R_l)
11
12             R_l ← realise(c, T_w) ∪ R_l
13             C_l ← C_l \ {c}
14         end
15         while C_r ≠ {} do
               c ← arg max_{c∈C_r} prob_ngram(R_r ∪ LMO(c, T_w))
16
17             R_r ← R_r ∪ realise(c, T_w)
18             C_r ← C_r \ {c}
19         end
20         return R_l ∪ R_r
21     end
22 end
```

Algorithm 2: The Language Model Ordering algorithm for linearising an T_w.

where $k = min(n-1, j-1)$, and,

$$prob_{MLE}(w_{i+k}|w_i \dots w_{i+k-1}) = \frac{cnt(w_i \dots w_{i+k})}{cnt(w_i \dots w_{i+k-1})} \quad (4)$$

where $prob_{MLE}(w_{i+k}|w_i \dots w_{i+k-1})$ is the maximum likelihood estimate *n*-gram probability. We refer to this tree linearisation method as the *Language Model Ordering* (LMO).

4 Using an Argument Satisfaction Model

4.1 Assigning Words to Argument Positions

One limitation of using the CLE algorithm for generation is that the resulting tree, though maximal in probability, may not conform to basic linguistic properties of a dependency tree. In particular, it may not have the correct number of arguments for each head word. That is, a word may have too few or too many modifiers.

To address this problem, we can take into account the argument position when assigning a weight to an edge. When attaching an edge connecting a modifier to a head to the spanning tree, we count how many modifiers the head already has. An edge is penalised if it is improbable that the head takes on yet another modifier, say in the example of an attachment to a preposition whose argument position has already been filled.

However, accounting for argument positions makes an edge weight dynamic and dependent on surrounding tree context. This makes the search for an optimal tree an NP-hard problem (McDonald and Satta, 2007) as all possible trees must be considered to find an optimal solution.

Consequently, we must choose a heuristic search algorithm for finding the locally optimum spanning tree. By representing argument positions that can be filled only once, we allow modifiers to compete for argument positions and vice versa. The CLE algorithm only considers this competition in one direction. In line 3 of Algorithm 1, only heads compete for modifiers, and thus the solution will be sub-optimal. In Wan et al. (2007), we showed that introducing a model of argument positions into a greedy spanning tree algorithm had little effect on performance. Thus, to consider both directions of competition, we design a new algorithm for constructing (dependency) spanning trees that casts edge selection as a weighted bipartite graph matching (or assignment) problem.

This problem is to find a weighted alignments between objects of two distinct sets, where an object from one set is uniquely aligned to some object in the other set. The optimal alignment is one where the sum of alignment costs is minimal. The graph of all possible assignments is a weighted bipartite graph. Here, to discuss bipartite graphs, we will extend our notation in a fairly standard way, to write $G^p = (U, V, E^p)$, where U, V are the disjoint sets of vertices and E^p the set of edges.

In our paper, we treat the assignment between attachment positions and words as an assignment problem. The standard polynomial-time solution to the assignment problem is the Kuhn-Munkres (or Hungarian) algorithm (Kuhn, 1955).[2]

4.2 A Dependency-Spanning Tree Algorithm

Our alternative dependency-spanning tree algorithm, presented as Algorithm 3, incrementally adds vertices to a growing spanning tree. At each iteration, the Kuhn-Munkres method assigns words that are as yet unattached to argument positions already available in the tree. We focus on the bipartite graph in Section 4.3.

Let the sentence w have the dependency graph $G_w = (V_w, E_w)$. At some arbitrary iteration of the algorithm (see Figure 1), we have the following:

- $T_w \subseteq E_w$, the set of edges in the spanning tree constructed so far;

[2] GPL code: `http://sites.google.com/site/garybaker/hungarian-algorithm/assignment`

Partially determined spanning tree:

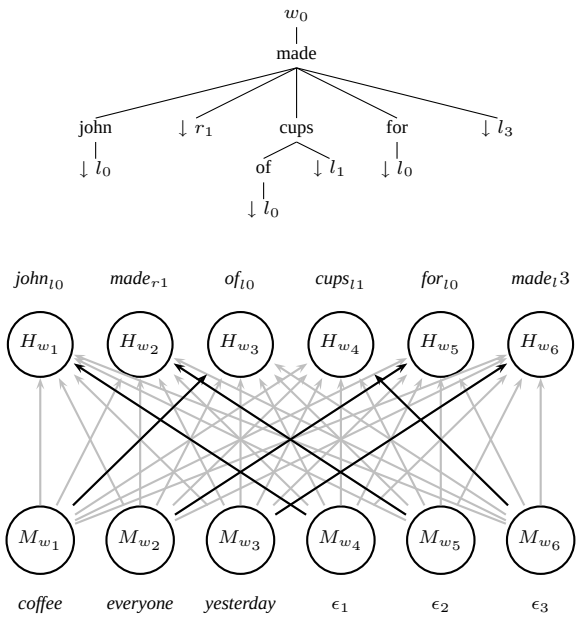

Figure 1: A snapshot of the generation process. Each word in the tree has argument positions to which we can assign remaining words. Padding M_w with ϵ is described in Section 4.3.

- $H_w = \{u, v \,|\, (u, v) \in T_w\}$, the set of vertices in T_w, or 'attached vertices', and therefore potential heads; and
- $M_w = V_w \backslash H_w$, the set of 'unattached vertices', and therefore potential modifiers.

For the potential heads, we want to define the set of possible attachment positions available in the spanning tree where the potential modifiers can attach. To talk about these attachment positions, we define the set of labels $\mathcal{L} = \{(d, j) | d \in D, j \in \mathbb{N}\}$, an element (d, j) representing an attachment point in direction d, position j. Valid attachment positions must be in sequential order and not missing any intermediate positions (e.g. if position 2 on the right is specified, position 1 must be also): so we define for some $i \in \mathbb{N}$, $0 \leq i < N$, a set $A_i \subseteq \mathcal{L}$ such that if the label $(d, j) \in A_i$ then the label $(d, k) \in A_i$ for $0 \leq k < j$. Collecting these, we define $A = \{A_i \,|\, 0 \leq i < N\}$.

To map a potential head onto the set of attachment positions, we define a function $q : H_w \rightarrow A$. So, given some $v \in H_w$, $q(v) = A_i$ for some $0 \leq i < N$. In talking about an individual attachment point $(d, j) \in q(v)$ for potential head v, we use the notation v_{dj}. For example, when referring to the second argument position on the right with respect to v, we use v_{r2}.

```
   /* initialisation */
 1 H_w ← {w_0}
 2 M_w ← V'
 3 U_w ← {w_0_R1}
 4 U'_w ← {}
 5 T_w ← {}
   /* The Assignment-based Algorithm */
 6 begin
 7     while M_w ≠ {} and U'_w ≠ U_w do
 8         U'_w ← U_w
 9         foreach ⟨u, (d, j)), v⟩ ∈ Kuhn-Munkres(G^p_w =
           (U_w, M^ε_w, E^p_w)) do
10             T_w ← T_w ∪ {(u, v)}
11             if u ∈ H_w then
12                 U_w ← U_w \ {u}
13             end
14             U_w ← U_w ∪ next(q(u))
15             U_w ← U_w ∪ next(q(m))
16             q(m) ← q(m) \ next(q(m))
17             q(h) ← q(h) \ next(q(h))
18             M_w ← M_w \ {m}
19             H_w ← H_w ∪ {m}
20         end
21     end
22 end
```

Algorithm 3: The Assignment-based Dependency Tree Building algorithm.

For the implementation of the algorithm, we have defined q, to specify attachment points, as follows, given some $v \in H_w$:

$$q(v) = \begin{cases} \{v_{r1}\} & \text{if } v = w_0, \text{the root} \\ \{v_{l1}\} & \text{if pos}(v) \text{ is a preposition} \\ \mathcal{L} & \text{if pos}(v) \text{ is a verb} \\ \{v_{lj} \,|\, j \in \mathbb{N}\} & \text{otherwise} \end{cases}$$

Defining q allows one to optionally incorporate linguistic information if desired.

We define the function *next* : $q(v) \rightarrow A, v \in H_w$ that returns the position (d, j) with the smallest value of j for direction d. Finally, we write the set of available attachment positions in the spanning tree as $U \subseteq \{(v, l) \,|\, v \in H_w, l \in q(v)\}$.

4.3 Finding an Assignment

To construct the bipartite graph used for the assignment problem at line 9 of Algorithm 3, given our original dependency graph $G_w = (V_w, E_w)$, and the variables defined from it above in Section 4.2, we do the following. The first set of vertices, of possible heads and their attachment points, is the set U_w. The second set of vertices is the set of possible modifiers augmented by dummy vertices ϵ_i (indicating no modification) such that this set is at least as large as U_w : $M^\epsilon_w = M_w \cup \{\epsilon_0, \ldots, \epsilon_{max(0, |U_w| - |M_w|)}\}$. The bi-

partite graph is then $G^p_w = (U_w, M^\epsilon_w, E^p_w)$, where $E^p_w = \{(u, v) \mid u \in U_w, v \in M^\epsilon_w\}$.

The weights on the edges of this graph incorporate a model of argument counts. The weight function is of the form $s_{ap} : E^p \rightarrow \mathbb{R}$. We consider some $e \in E^p_w$: $e = (v', v)$ for some $v' \in U_w, v \in M^\epsilon_w$; and $v' = (u, (d, j))$ for some $u \in V_w, d \in \mathcal{D}, j \in \mathbb{N}$. $s(u, M^\epsilon_w)$ is defined to return the maximum cost so that the dummy leaves are only attached as a last resort. We then define:

$$s_{ap}(e) = -log(prob_{dep}(u, v, d) \times prob_{arg}(u, d, j)) \quad (5)$$

where $prob_{dep}(u, v, d)$ is as in equation 2, using the original dependency graph defined in Section 2; and $prob_{arg}(u, d, j)$, an estimate of the probability that a word u with i arguments assigned already can take on more arguments, is defined as:

$$prob_{arg}(u, d, j) = \frac{\sum_{i=j+1}^{\infty} cnt_{arg}(u, d, i)}{cnt(u, d)} \quad (6)$$

where $cnt_{arg}(u, d, i)$ is the number of times word u has been seen with i arguments in direction d; and $cnt(u, d) = \sum_{i \in \mathbb{N}} cnt_{arg}(u, d, i)$. As the probability of argument positions beyond a certain value for i in a given direction will be extremely small, we approximate this sum by calculating the probability density up to a fixed maximum, in this case 7 argument positions, and assume zero probability beyond that.

5 Evaluation

5.1 String Generation Task

The best-performing word ordering algorithm is one that makes fewest grammatical errors. As a surrogate measurement of grammaticality, we use the string regeneration task. Beginning with a human-authored sentence with its word order randomised, the goal is to regenerate the original sentence. Success is indicated by the proportion of the original sentence regenerated, as measured by any string comparison method: in our case, using the BLEU metric (Papineni et al., 2002). One benefit to this evaluation is that content selection, as a factor, is held constant. Specifically, the probability of word selection is uniform for all words.

The string comparison task and its associated metrics like BLEU are not perfect.[3] The evaluation can be seen as being overly strict. It assumes that the *only* grammatical order is that of the original human authored sentence, referred to as the 'gold standard' sentence. Should an approach chance upon an alternative grammatical ordering, it would penalised. However, all algorithms and baselines compared would suffer equally in this respect, and so this will be less problematic when averaging across multiple test cases.

5.2 Data Sets and Training Procedures

The Penn Treebank corpus (PTB) was used to provide a model of dependency relations and argument counts. It contains about 3 million words of text from the Wall Street Journal (WSJ) with human annotations of syntactic structures. Dependency events were sourced from the events file of the Collins parser package, which contains 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 4-gram language model (LM) was also obtained from the PTB training data, referred to as PTB-LM. Additionally, a 4-gram language model was obtained from a subsection of the BLLIP'99 Corpus (LDC number: LDC2000T43) containing three years of WSJ data from 1987 to 1989 (Charniak et al., 1999). As in Collins et al. (2004), the 1987 portion of the BLLIP corpus containing 20 million words was also used to create a language model, referred to here as BLLIP-LM. *N*-gram models were smoothed using Katz's method, backing off to smaller values of n.

For this evaluation, tokenisation was based on that provided by the PTB data set. This data set also delimits base noun phrases (noun phrases without nested constituents). Base noun phrases were treated as single tokens, and the rightmost word assumed to be the head. For the algorithms tested, the input set for any test case consisted of the single tokens identified by the PTB tokenisation. Additionally, the heads of base noun phrases were included in this input set. That is, we do not regenerate the base noun phrases.[4]

[3] Alternative grammaticality measures have been developed recently (Mutton et al., 2007). We are currently exploring the use of this and other metrics.

[4] This would correspond to the use of a chunking algorithm or a named-entity recogniser to find noun phrases that could be re-used for sentence generation.

Algorithms	PTB-LM	BLLIP-LM
Viterbi baseline	14.9	18.0
LMO baseline	24.3	26.0
CLE	26.4	26.8
AB	33.6	33.7

Figure 2: String regeneration as measured in BLEU points (maximum 100)

5.3 Algorithms and Baselines

We compare the baselines against the Chu-Liu Edmonds (CLE) algorithm to see if spanning tree algorithms do indeed improve upon conventional language modelling. We also compare the Assignment-based (AB) algorithm against the baselines and CLE to see if, additionally, modelling argument assignments improves the resulting tree and thus the generated word sequence. Two baseline generators based on n-gram language-models were used, representing approaches that optimise word sequences based on the local context of the n-grams.

The first baseline re-uses the *LMO* greedy sequence algorithm on the same set of input words presented to the CLE and AB algorithms. We apply LMO in a rightward manner beginning with a start-of-sentence token. Note that this baseline generator, like the two spanning tree algorithms, will score favourably using BLEU since, minimally, the word order of the base noun phrases will be correct when each is reinserted.

Since the LMO baseline reduces to bigram generation when concatenating single words, we test a second language model baseline which always uses a 4-gram window size. A Viterbi-like generator with a 4-gram model and a beam of 100 is used to generate a sequence. For this baseline, referred to as the *Viterbi* baseline, base noun phrases were separated into their constituent words and included in the input word set.

5.4 Results

The results are presented in Table 2. Significance was measured using the sign test and the sampling method outlined in (Collins et al., 2005). We will examine the results in the PTB-LM column first. The gain of 10 BLEU points by the LMO baseline over the Viterbi baseline shows the performance improvement that can be gained when reinserting the base noun phrases.

AB: the dow at this point was down about 35 points
CLE: was down about this point 35 points the dow at
LMO: was this point about at down the down 35 points
Viterbi: the down 35 points at was about this point down
Original: at this point, the dow was down about 35 points

Figure 3: Example generated sentences using the BLLIP-LM.

The CLE algorithm significantly out-performed the LMO baseline by 2 BLEU points, from which we conclude that incorporating a model for global syntactic structure and treating the search for a dependency tree as a spanning problem helps for novel sentence generation. However, the real improvement can be seen in the performance of the AB system which significantly out-performs all other methods, beating the CLE algorithm by 7 BLEU points, illustrating the benefits of a model for argument counts and of couching tree building as an iterative set of argument assignments.

One might reasonably ask if more *n*-gram data would narrow the gap between the tree algorithms and the baselines, which encode global and local information respectively. Examining results in the BLLIP-LM column, all approaches improve with the better language model. Unsurprisingly, the improvements are most evident in the baselines which rely heavily on the language model. The margin narrows between the CLE algorithm and the LMO baseline. However, the AB algorithm still out-performs all other approaches by 7 BLEU points, highlighting the benefit in modelling dependency relations. Even with a language model that is one order of magnitude larger than the PTB-LM, the AB still maintains a sizeable lead in performance. Figure 3 presents sample generated strings.

6 Related Work

6.1 Statistical Surface Realisers

The work in this paper is similar to research in statistical surface realisation (for example, Langkilde and Knight (1998); Bangalore and Rambow (2000); Filippova and Strube (2008)). These start with a semantic representation for which a specific rendering, an ordering of words, must be determined, often using language models to govern tree traversal. The task in this paper is different as it is a text-to-text scenario and does not begin with a representation of semantics.

The dependency model and the LMO linearisation algorithm are based heavily on word order statistics. As such, the utility of this approach is limited to human languages with minimal use of inflections, such as English. Approaches for other language types, for example German, have been explored (Filippova and Strube, 2007).

6.2 Text-to-Text Generation

As a text-to-text approach, our work is more similar to work on Information Fusion (Barzilay et al., 1999), a sub-problem in multi-document summarisation. In this work, sentences presenting the same information, for example multiple news articles describing the same event, are merged to form a single summary by aligning repeated words and phrases across sentences.

Other text-to-text approaches for generating novel sentences also aim to recycle sentence fragments where possible, as we do. Work on phrase-based statistical machine translation has been applied to paraphrase generation (Bannard and Callison-Burch, 2005) and multi-sentence alignment in summarisation (Daumé III and Marcu, 2004). These approaches typically use *n*-gram models to find the best word sequence.

The WIDL formalism (Soricut and Marcu, 2005) was proposed to efficiently encode constraints that restricted possible word sequences, for example dependency information. Though similar, our work here does not explicitly represent the word lattice.

For these text-to-text systems, the order of elements in the generated sentence is heavily based on the original order of words and phrases in the input sentences from which lattices are built. Our approach has the benefit of considering all possible orderings of words, corresponding to a wider range of paraphrases, provided with a suitable dependency model is available.

6.3 Parsing and Semantic Role Labelling

This paper presents work closely related to parsing work by McDonald et al. (2005) which searches for the best parse tree. Our work can be thought of as generating projective dependency trees (that is, without crossing dependencies).

The key difference between parsing and generation is that, in parsing, the word order is fixed, whereas for generation, this must be determined. In this paper, we search across all possible tree structures whilst searching for the best word ordering. As a result, an argument model is needed to identify linguistically plausible spanning trees.

We treated the alignment of modifiers to head words as a bipartite graph matching problem. This is similar to work in semantic role labelling by Padó and Lapata (2006). The alignment of answers to question types as a semantic role labelling task using similar methods was explored by Shen and Lapata (2007).

Our work is also strongly related to that of Wong and Mooney (2007) which constructs symbolic semantic structures via an assignment process in order to provide surface realisers with input. Our approach differs in that we do not begin with a fixed set of semantic labels. Additionally, our end goal is a dependency tree that encodes word precedence order, bypassing the surface realisation stage.

7 Conclusions

In this paper, we presented a new use of spanning tree algorithms for generating sentences from an input set of words, a task common to many text-to-text scenarios. The algorithm finds the best dependency trees in order to ensure that the resulting string has grammaticality modelled at a global (sentence) level. Our algorithm incorporates a model of argument satisfaction which is treated as an assignment problem, using the Kuhn-Munkres assignment algorithm. We found a significant improvement using BLEU to measure improvements on the string regeneration task. We conclude that our new algorithm based on the assignment problem and an argument model finds trees that are linguistically more plausible, thereby improving the grammaticality of the generated word sequence.

References

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

Srinivas Bangalore, Owen Rambow, and Steve Whittaker. 2000. Evaluation metrics for generation. In *Proceedings of the first international conference on Natural language generation*, Morristown, NJ, USA.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Asso-*

ciation 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.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 1999. Bllip 1987-89 wsj corpus release 1. Technical report, Linguistic Data Consortium.

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, v.14:1396–1400.

Christopher Collins, Bob Carpenter, and Gerald Penn. 2004. Head-driven parsing for word lattices. In *Proceedings of the 42nd Annual Meeting 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 Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, San Francisco.

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

J. Edmonds. 1967. Optimum branchings. *J. Research of the National Bureau of Standards*, 71B:233–240.

Katja Filippova and Michael Strube. 2007. Generating constituent order in german clauses. In *Proceedings of the 45th Annual Meeting on Association for Computational Linguistics*. Prague, Czech Republic.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Conference on Empirical Methods in Natural Language Processing*, Waikiki, Honolulu, Hawaii.

Mark Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable cfgs with unfold-fold. In *Proceedings of the 45th Annual Meeting on Association for Computational Linguistics*. Prague, Czech Republic.

H.W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 219552:83–97 83–97.

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.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, Prague, Czech Republic.

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.

Sebastian Padó and Mirella Lapata. 2006. Optimal constituent alignment with edge covers for semantic projection. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, Morristown, NJ, USA.

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, July.

Dan Shen and Mirella Lapata. 2007. Using semantic roles to improve question answering. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic.

Radu Soricut and Daniel Marcu. 2005. Towards developing generation algorithms for text-to-text applications. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, Michigan.

Stephen Wan, Robert Dale, Mark Dras, and Cécile Paris. 2007. Global revision in summarisation: Generating novel sentences with prim's algorithm. In *Proceedings of 10th Conference of the Pacific Association for Computational Linguistic*, Melbourne, Australia.

Yuk Wah Wong and Raymond Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, Rochester, New York.