

# Efficient extraction of grammatical relations

Rebecca Watson, John Carroll<sup>†</sup> and Ted Briscoe

Computer Laboratory, University of Cambridge, Cambridge, CB3 0FD, UK

`firstname.lastname@cl.cam.ac.uk`

<sup>†</sup>Department of Informatics, University of Sussex, Brighton BN1 9QH, UK

`J.A.Carroll@sussex.ac.uk`

## Abstract

We present a novel approach for applying the Inside-Outside Algorithm to a packed parse forest produced by a unification-based parser. The approach allows a node in the forest to be assigned multiple inside and outside probabilities, enabling a set of ‘weighted GRs’ to be computed directly from the forest. The approach improves on previous work which either loses efficiency by unpacking the parse forest before extracting weighted GRs, or places extra constraints on which nodes can be packed, leading to less compact forests. Our experiments demonstrate substantial increases in parser accuracy and throughput for weighted GR output.

## 1 Introduction

RASP is a robust statistical analysis system for English developed by Briscoe and Carroll (2002). It contains a syntactic parser which can output analyses in a number of formats, including (n-best) syntactic trees, robust minimal recursion semantics (Copestake, 2003), grammatical relations (GRs), and weighted GRs. The weighted GRs for a sentence comprise the set of grammatical relations in all parses licensed for that sentence, each GR is weighted based on the probabilities of the parses in which it occurs. This weight is normalised to fall within the range  $[0,1]$  where 1.0 indicates that all parses contain the GR. Therefore, high precision GR sets can be determined by thresholding on the GR weight (Carroll and Briscoe, 2002). Carroll and

Briscoe compute weighted GRs by first unpacking all parses or the n-best subset from the parse forest. Hence, this approach is either (a) inefficient (and for some examples impracticable) if a large number of parses are licensed by the grammar, or (b) inaccurate if the number of parses unpacked is less than the number licensed by the grammar.

In this paper, we show how to obviate the need to trade off efficiency and accuracy by extracting weighted GRs directly from the parse forest using a dynamic programming approach based on the Inside-Outside algorithm (IOA) (Baker, 1979; Lari and Young, 1990). This approach enables efficient calculation of weighted GRs over *all parses* and substantially improves the throughput and memory usage of the parser. Since the parser is unification-based, we also modify the parsing algorithm so that local ambiguity packing is based on feature structure equivalence rather than subsumption.

Similar dynamic programming techniques that are variants of the IOA have been applied for related tasks, such as parse selection (Johnson, 2001; Schmid and Rooth, 2001; Geman and Johnson, 2002; Miyao and Tsujii, 2002; Kaplan et al., 2004; Taskar et al., 2004). The approach we take is similar to Schmid and Rooth’s (2001) adaptation of the algorithm, where ‘expected governors’ (similar to our ‘GR specifications’) are determined for each tree, and alternative nodes in the parse forest have the *same lexical head*. Initially, they create a packed parse forest and during a second pass the parse forest nodes are split if multiple lexical heads occur. The IOA is applied over this split data structure. Similarly, Clark and Curran (2004) alter their packing algorithm so that nodes in the packed chart have the same semantic head and ‘unfilled’ GRs. Our ap-

proach is novel in that while calculating inside probabilities we allow any node in the parse forest to have *multiple semantic heads*.

Clark and Curran (2004) apply Miyao and Tsujii's (2002) dynamic programming approach to determine weighted GRs. They outline an alternative parse selection method based on the resulting weighted GRs: select the (consistent) GR set with the highest average weighted GR score. We apply this parse selection approach and achieve 3.01% relative reduction in error. Further, the GR set output by this approach is a *consistent set* whereas the high precision GR sets outlined in (Carroll and Briscoe, 2002) are neither consistent nor coherent.

The remainder of this paper is organised as follows: Section 2 gives details of the RASP system that are relevant to this work. Section 3 describes our test suite and experimental environment. Changes required to the current parse forest creation algorithm are discussed in Section 4, while Section 5 outlines our dynamic programming approach for extracting weighted GRs (EWG). Section 6 presents experimental results showing (a) improved efficiency achieved by EWG, (b) increased upper bounds of precision and recall achieved using EWG, and (c) increased accuracy achieved by a parse selection algorithm that would otherwise be too inefficient to consider. Finally, Section 7 outlines our conclusions and future lines of research.

## 2 The RASP System

RASP is based on a pipelined modular architecture in which text is pre-processed by a series of components including sentence boundary detection, tokenisation, part of speech tagging, named entity recognition and morphological analysis, before being passed to a statistical parser<sup>1</sup>. A brief overview of relevant aspects of syntactic processing in RASP is given below; for full details of system components, see Briscoe and Carroll (1995; 2002; 2005)<sup>2</sup>.

---

<sup>1</sup>Processing times given in this paper do not include these pre-processing stages, since they take negligible time compared with parsing.

<sup>2</sup>RASP is freely available for research use; visit <http://www.informatics.susx.ac.uk/research/nlp/rasp/>

## 2.1 The Grammar

Briscoe and Carroll (2005) describe the (manually-written) feature-based unification grammar and the rule-to-rule mapping from local trees to GRs. The mapping specifies for each grammar rule the semantic head(s) of the rule (henceforth, head), and one or more GRs that should be output (optionally depending on feature values instantiated at parse time). For example, Figure 1 shows a grammar rule analysing a verb phrase followed by a prepositional phrase modifier. The rule identifies the first daughter (1) as the semantic head, and specifies that one of five possible GRs is to be output, depending on the value of the `PSUBCAT` syntactic feature; so, for example, if the feature has the value `NP`, then the relation is `nmod` (non-clausal modifier), with slots filled by the semantic heads of the first and second daughters (the 1 and 2 arguments).

Before parsing, a context free backbone is derived automatically from the grammar, and an LALR(1) parse table is computed from this backbone (Carroll, 1993, describes the procedure in detail). Probabilities are associated with actions in the parse table, by training on around 4K sentences from the Susanne corpus (Sampson, 1995), each sentence having been semi-automatically converted from a tree-bank bracketing to a tree conforming to the unification grammar (Briscoe and Carroll, 1995).

## 2.2 The Parse Forest

When parsing, the LALR table action probabilities are used to assign a score to each newly derived (sub-)analysis. Additionally, on each reduce action (i.e. complete application of a rule), the rule's daughters are unified with the sequence of sub-analyses being consumed. If unification fails then the reduce action is aborted. Local ambiguity packing (packing, henceforth) is performed on the basis of feature structure subsumption. Thus, the parser builds and returns a compact structure that efficiently represents all parses licensed by the grammar: the parse forest. Since unification often fails it is not possible to apply beam or best first search strategies during construction of the parse forest; statistically high scoring paths often end up in unification failure. Hence, the parse forest represents all parses licensed by the grammar.

```

V1/vp_pp : V1[MOD +] --> H1 P2[ADJ -, WH -] :
  1 :
  2 = [PSUBCAT NP], (ncmod _ 1 2) :
  2 = [PSUBCAT NONE], (ncmod prt 1 2) :
  2 = [PSUBCAT (VP, VPINF, VPING, VPPRT, AP)], (xmod _ 1 2) :
  2 = [PSUBCAT (SFIN, SINF, SING)], (cmod _ 1 2) :
  2 = [PSUBCAT PP], (pmod 1 2).

```

Figure 1: Example grammar rule, showing the rule name and syntactic specification (on the first line), identification of daughter 1 as the semantic head (second line), and possible GR outputs depending on the parse-time value of the `PSUBCAT` feature of daughter 2 (subsequent lines).

Figure 2 shows a simplified parse forest containing three parses generated for the following pre-processed text<sup>3</sup>:

```

I_PPIS1 see+ed_VVD the_AT man_NN1
in_II the_AT park_NN1

```

The GR specifications shown are instantiated based on the values of syntactic features at daughter nodes, as discussed in Section 2.1 above. For example, the `V1/vp_pp` sub-analysis (towards the left hand side of the Figure) contains the instantiated GR specification `<1, (ncmod _ 1 2)>` since its second daughter has the value `NP` for its `PSUBCAT` feature.

Henceforth, we will use the term ‘node’ to refer to data structures in our parse forest corresponding to a rule instantiation: a sub-analysis resulting from application of a reduce action. Back pointers are stored in nodes, indicating which daughters were used to create the sub-analysis. These pointers provide a means to traverse the parse forest during subsequent processing stages. A ‘packed node’ is a node representing a sub-analysis that is subsumed by, and hence packed into, another node. Packing is considered for nodes only if they are produced in the same LR state and represent sub-analyses with the same word span. A parse forest can have a number of root nodes, each one dominating analyses spanning the whole sentence with the specified top category.

### 2.3 Parser Output

From the parse forest, RASP unpacks the ‘n-best’<sup>4</sup> syntactic trees using a depth-first beam search (Carroll, 1993). There are a number of types of analysis

<sup>3</sup>The part of speech tagger uses a subset of the Lancaster CLAWS2 tagset – <http://www.comp.lancs.ac.uk/computing/research/ucrel/claws2tags.html>

<sup>4</sup>This number  $n$  is specified by the user, and represents the maximal number of parses to be unpacked.

output available, including syntactic tree, grammatical relations (GRs) and robust minimal recursion semantics (RMRS). Each of these is computed from the n-best trees.

Another output possibility is weighted GRs (Carroll and Briscoe, 2002); this is the unique set of GRs from the n-best GRs, each GR weighted according to the sum of the probabilities of the parses in which it occurs. Therefore, a number of processing stages determine this output: unpacking the n-best syntactic trees, determining the corresponding n-best GR sets and finding the unique set of GRs and corresponding weights.

The GRs for each parse are computed from the set of GR specifications at each node, passing the (semantic) head of each sub-analysis up to the next higher level in the parse tree (beginning from word nodes). GR specifications for nodes (which, if required, have been instantiated based on the features of daughter nodes) are referred to as ‘unfilled’ until the slots containing numbers are ‘filled’ with the corresponding heads of daughter nodes. For example, the grammar rule named `NP/det_n` has the unfilled GR specification `<2, (det 2 1)>`. Therefore, if an `NP/det_n` local tree has two daughters with heads *the* and *cat* respectively, the resulting filled GR specification will be `<cat, (det cat the)>`, i.e. the head of the local tree is *cat* and the GR output is `(det cat the)`.

Figure 3 illustrates the n-best GRs and the corresponding (non-normalised and normalised) weighted GRs for the sentence *I saw the man in the park*. The corresponding parse forest for this example is shown in Figure 2. Weights on the GRs are normalised probabilities representing the weighted proportion of parses in which the GR occurs. This weighting is in practice calculated as the sum of parse probabilities for parses con-



taining the specific GR, normalised by the sum of all parse probabilities. For example, the GR (iobj see+ed in) is in one parse with probability  $-28.056154$ , the non-normalised score. The sum of all parse probabilities is  $-28.0200896$ . Therefore, the normalised probability (and final weight) of the GR is  $10^{(-28.056154 - (-28.0200896))} = 0.920314^5$ .

### 3 Data and Methods

King et al. (2003) outline the development of the PARC 700 Dependency Bank (henceforth, DepBank), a gold-standard set of relational dependencies for 700 sentences (originally from the Wall Street Journal) drawn at random from Section 23 of the Penn Treebank. Briscoe and Carroll (2005) extended DepBank with a set of gold-standard RASP GRs that we use to measure parser accuracy.

We use the same 560 sentence subset from the DepBank utilised by Kaplan et al. (2004) in their study of parser accuracy and efficiency. All experimental results are obtained using this test suite on an AMD Opteron 2.5GHz CPU with 1GB of Ram on a 64 bit version of Linux. The parser’s output is evaluated using a relational dependency evaluation scheme (Carroll et al., 1998; Lin, 1998) and standard evaluation measures: precision, recall and  $F_1$ .

### 4 Local Ambiguity Packing

Open and Carroll (2000) note that when using subsumption-based packing with a unification-based grammar, the parse forest may implicitly represent some parses that are not actually licensed by the grammar; these will have values for one or more features that are locally but not globally consistent. This is not a problem when computing GRs from trees that have already been unpacked, since the relevant unifications will have been checked during the unpacking process, and will have caused the affected trees to be filtered out. Unification fails for at least one packed tree in approximately 10% of the sentences in the test suite. However, such inconsistent

<sup>5</sup>As we are dealing with log probabilities, summation and subtraction of these probabilities is not straightforward. Multiplication of probabilities  $X$  and  $Y$ , with log probabilities  $x$  and  $y$  respectively is determined using the formula  $X \times Y = x + y$ , division using  $X \div Y = x - y$ , summation using  $X + Y = x + \log_{10}(1 + 10^{(y-x)})$  and subtraction using  $X - Y = x + \log_{10}(1 - 10^{(y-x)})$ .

trees are a problem for any approach to probability computation over the parse forest that is based on the Inside-Outside algorithm (IOA). For our efficient weighted GR extraction technique we therefore modify the parsing algorithm so that packing is based on feature structure *equality* rather than subsumption.

Open and Carroll give definitions and implementation details for subsumption and equality operations, which we adopt. In the experiments below, we refer to versions of the parser with subsumption and equality based packing as SUB-PACKING and EQ-PACKING respectively.

### 5 Extracting Weighted GRs

Parse forest unpacking consumes larger amounts of CPU time and memory as the number of parses to unpack (n-best) increases. Carroll and Briscoe (2002) demonstrate that increasing the size of the n-best list increases the upper bound on precision (i.e. when low-weighted GRs are filtered out). Therefore, if practicable, it is preferable to include all possible parses when calculating weighted GRs. We describe below a dynamic programming approach (EWG) based on the IOA to efficiently extract weighted GRs directly from the parse forest. EWG calculates weighted GRs over all parses represented in the parse forest.

Inside and outside probabilities are analogous to the forward and backward probabilities of markov model algorithms. The inside probability represents the probability of all possible sub-analyses of a node. Conversely, the outside probability represents the probability of all analyses for which the node is a sub-analysis.

The IOA is ideal for our task, as the product of inside and outside probabilities for a sub-analysis constitutes part of the sum for the non-normalised weight of each GR (arising from the GR specification in the sub-analysis). Further, we can apply the sum of inside probabilities for each root-node, to normalise the weighted GRs.

#### 5.1 Implementation

Three processing stages are required to determine weighted GRs over the parse forest, calculating (1) filled GRs and corresponding inside probabili-

N-BEST GRS	(NON NORMALISED) WEIGHTED GRS
parse (log) probability: -28.056154	-28.0201 (ncsubj see+ed_VVD I_PPIS1 _)
(ncsubj see+ed I _)	-35.1598 (ncmod _ man_NN1 in_II)
(iobj see+ed in)	-28.0201 (det park_NN1 the_AT)
(dobj see+ed man)	-29.1187 (ncmod _ see+ed_VVD in_II)
(dobj in park)	-28.0562 (iobj see+ed_VVD in_II)
(det park the)	-28.0201 (dobj see+ed_VVD man_NN1)
(det man the)	-28.0201 (dobj in_II park_NN1)
	-28.0201 (det man_NN1 the_AT)
parse (log) probability: -29.11871	(NORMALISED) WEIGHTED GRS
(ncsubj see+ed I _)	
(ncmod _ see+ed in)	
(dobj in park)	1.0 (det park the)
(det park the)	1.0 (det man the)
(dobj see+ed man)	1.0 (dobj see+ed man)
(det man the)	1.0 (dobj in park)
	0.920314 (iobj see+ed in)
parse (log) probability: -35.159805	7.249102e-8 (ncmod _ man in)
(ncsubj see+ed I _)	7.968584e-2 (ncmod _ see+ed in)
(dobj see+ed man)	1.0 (ncsubj see+ed I _)
(det man the)	
(ncmod _ man in)	
(dobj in park)	
(det park the)	
Total Probability (log-sum of all parses): -28.0200896	

Figure 3: The n-best GRs, and non-normalised/normalised weighted GRs determined from three parses for the sentence *I saw the man in the park*. Parse probabilities and non-normalised weights are shown as log probabilities. Weights and parse probabilities are shown with differing precision, however RASP stores all probabilities in log (base 10) form with double float precision.

ties, (2) outside (and non-normalised) probabilities of weighted GRs, and (3) normalised probabilities of weighted GRs.<sup>6</sup> The first two processing stages are covered in detail in the following sections, while the final stage simply entails normalising the probabilities by dividing each weight by the sum of all the parse probabilities (the sum of root-nodes’ inside probabilities).

### 5.1.1 Inside probability and GR

To determine inside probabilities over the nodes in the parse forest, we need to propagate the head and corresponding inside probability upwards after filling the node’s GR specification. The inside probability of node  $n$  is usually calculated over the parse forest by multiplying the inside probability of the node’s daughters and the probability  $r(n)$  of the node itself (i.e. the probability of the shift or reduce action that caused the node to be created). Therefore, if a node has daughters  $D_1$  and  $D_2$ , then the inside probability  $E_n$  is calculated using:

$$E_n = E_{D_1} \times E_{D_2} \times r(n) \quad (1)$$

However, packed nodes each correspond to an alternative filled GR specification. Inside probabilities for these GR specifications need to be combined. If packed analyses  $n_p$  occur in node  $n$  then the inside probability of node  $n$  is:

$$E_n = \sum_{i \in (n, n_p)} E_i \quad (2)$$

Further, the alternative GR specifications may not necessarily specify the same head as the node’s GR specification and *multiple heads* may be passed up by the node. Hence, the summation in equation 2 needs to be conditioned on the possible heads of a node  $H(node)$ , where  $E_n^h$  is the inside probability of each head  $h$  for node  $n$ :

$$E_n^h = \sum_{i \in (n, n_p), H(i)=h} E_i \quad (3)$$

When multiple heads are passed up by daughter nodes, *multiple filled GR specifications* are found for the node. We create one filled GR specification for

<sup>6</sup>Note that the IOA is not applied iteratively; a single pass only is required.

each possible combination of daughters’ heads<sup>7</sup>. For example, consider the case where a node has daughters  $D_1$  and  $D_2$  with semantic heads  $\{dog, cat\}$  and  $\{an\}$  respectively. Here, we need to fill the GR specification  $\langle 2, (\det 2 1) \rangle$  with two sets of daughters’ heads:  $\langle dog, (\det dog an) \rangle$  and  $\langle cat, (\det cat an) \rangle$ .

As a node can have multiple filled GR specifications  $GR(node)$ , we alter equation 3 to:

$$E_n^h = \sum_{i \in (n, n_p)} \left( \sum_{j \in GR(i), H(j)=h} E_j \right) \quad (4)$$

Here,  $E_j$  (the inside probability of filled GR specification  $j$ ) is determined by multiplying the inside probabilities of daughters’ heads (that filled the GR specification) and the reduce probability of the node itself, i.e. using a modification of equation 1. Returning to the previous example, the inside probabilities of  $\langle dog, (\det dog an) \rangle$  and  $\langle cat, (\det cat an) \rangle$  will be equal to the reduce probability of the node multiplied by (a) the inside probability of head  $an$ , and (b) the inside probabilities of the heads  $dog$  and  $cat$ , respectively.

Hence, (a) calculation of inside probabilities takes into account multiple semantic heads, and (b) GR specifications are filled using every possible combination of daughters’ heads. Each node  $n$  is processed in full as follows:

- Process each of the node’s packed nodes  $n_p$  to determine the packed node’s list of filled GR specifications and corresponding inside probabilities.
- Process the node  $n$ , with daughters  $D_n$ :
  - Instantiate  $n$ ’s GR specifications based on features of  $D_n$ .
  - Process each daughter in  $D_n$  to determine a list of possible semantic heads and corresponding inside probabilities for each.
  - Fill the GR specification of  $n$  with each possible combination of daughters’ heads.

<sup>7</sup>The same word can appear as a head for more than one daughter of a node. This occurs if competing analyses have daughters with different word spans and, therefore, particular words can be considered in the span of either daughter. As the grammar permits both pre- and post- modifiers, it is possible for words in the ‘overlapping’ span to be passed up as heads for both daughters. Therefore, semantic heads are not combined unless they are different words.

Calculate the inside probability of each filled GR specification.

- Combine the alternative filled GR specifications of  $n$  and  $n_p$ , determining the list of unique semantic heads and corresponding inside probabilities using equation 4.

For each node, we propagate up a set of data structures  $\{S_h\}$  that each contain one possible head  $h$  and corresponding inside probability. At word nodes, we simply return the word and the reduce score of the word as the semantic head and inside probability, respectively. Back pointers are also included to store the list of alternative filled GR specifications and corresponding inside probabilities, the reduce score for the node and the daughters’ data structures (used to fill the GR specifications).

### 5.1.2 Outside probability determination

After the inside probabilities have been computed (bottom-up) the resulting data structure at the root-node is traversed to compute outside probabilities. The data structure created is split into alternative semantic heads for each node and, therefore, traversal to determine outside probabilities is relatively trivial: the outside probability of a filled GR specification is equal to the outside probability of the corresponding unique head of the node. Therefore, once we have created the new data structure, outside probabilities for each node can be determined over this structure in the regular fashion.

We calculate the outside probabilities (top-down) and, when we find filled GR specifications, we incrementally store the non-normalised weight of each GR. Each data structure  $S_h$  for head  $h$ , with outside probability  $F_h$ , is processed in full as follows:

- Process each of the GR specifications  $GR(S_h)$ . For each  $j \in GR(S_h)$ :
  - Let  $F_j = F_h$  and calculate the probability of  $j$ ,  $I_j = E_j \times F_j$ .
  - Add  $I_j$  to the (non-normalised) probability for  $j$  (in a hash table).
  - Process the data structure for each child head in  $j$ ,  $C(j)$ . That is, the daughters’ heads that filled the GR specification (resulting in  $j$ ). For each  $k \in C(j)$ :

- \* Calculate the outside probability of  $k$  (using the reduce probability of the node  $r(n)$ , stored in the data structure  $S_h$ ):

$$F_k = F_j \times r(n) \times \prod_{i \in C(j), i \neq k} E_i \quad (5)$$

- \* Queue the data structure  $k$  and corresponding outside probability  $F_k$ .<sup>8</sup>

## 6 Experimentation

### 6.1 Efficiency and Accuracy

The dynamic programming algorithm outlined in Section 5, EWG, provides an efficient and accurate method of determining weighted GRs directly from the parse forest. Figures 5 and 6 compare the efficiency of EWG to the EQ-PACKING and SUB-PACKING methods in terms of CPU time and memory, respectively<sup>9</sup>. Note that EWG applies equality-based packing to ensure only parses licensed by the grammar are considered (see Section 4).

As the maximum number of (n-best) parses increases, EQ-PACKING requires more time and memory than SUB-PACKING. However, if we compare these systems with an n-best value of 1, the difference in time and memory is negligible, suggesting that it is the unpacking stage which is responsible for the decreased throughput. For EWG we are forced to use equality-based packing, but these results suggest that using equality is not hurting the throughput of EWG.

Both figures illustrate that the time and memory required by EWG are static because the algorithm considers all parses represented in the parse forest regardless of the value of n-best specified. Therefore, the ‘cross-over points’ are of particular interest: at which n-best value is EWG’s efficiency the same as that of the current system? This value is

<sup>8</sup>We apply a breadth first search (FIFO queue) to minimise multiple processing of shared data structures. If an outside probability is determined for a data structure already queued, then the probability is appended to the queued item. The steps are modified to enable multiple outside probabilities, i.e. summation over each outside probability when calculating  $I_j$  and  $F_k$ .

<sup>9</sup>CPU time and memory usage are as reported using the `time` function in Allegro Common Lisp 7.0 and do not include system start-up overheads or the time required for garbage collection.

approximately 580 and 100 for time and memory, respectively (comparing EWG to EQ-PACKING). Given that there are on average around 9000 parses per sentence in the test suite, these results indicate a substantial improvement in both efficiency and accuracy for weighted GR calculation. However, the median number of parses per sentence is around 50, suggesting that large parse numbers for a small subset of the test suite are skewing the arithmetic mean. Therefore, the complexity of this subset will significantly decrease throughput and EWG will improve efficiency for these sentences more so than for others.

The general relationship between sentence length and number of parses suggests that the EWG will be more beneficial for longer sentences. Figure 4 shows the distribution of number of parses over sentence length. The figure illustrates that the number of parses can not be reliably predicted from sentence length. Considering the cross-over points for time and memory, the number of sentences with more than 580 and 100 parses were 216 and 276, respectively. Thus, the EWG out-performs the current algorithm for around half of the sentences in the data set. The relative gain achieved reflects that a subset of sentences can significantly decrease throughput. Hence, the EWG is expected to improve the efficiency if a) longer sentences are present in the data set and b) n-best is set to a value greater than the cross-over point(s).

Upper bounds on precision and recall can be determined using weight thresholds over the GRs of 1.0 and 0.0, respectively<sup>10</sup>. Upper bounds of precision and recall provided by EWG are 79.57 and 82.02, respectively, giving an  $F_1$  upper bound of 81.22%. However, considering the top 100 parses only, we achieve upper bounds on precision and recall of 78.77% and 81.18% respectively, resulting in an  $F_1$  upper bound of 79.96%. Therefore, using EWG, we are able to achieve a relative increase of 6.29% for the  $F_1$  upper bound on the task. Similarly, Carroll and Briscoe (2002) demonstrate (on an earlier, different test suite) that increasing the number of parses (n-best) from 100 to 1000 increases precision of weighted GR sets from 89.59% to 90.24%,

<sup>10</sup>In fact, in these experiments we use a threshold of  $1 - \epsilon$  (with  $\epsilon = 0.0001$ ) instead of a threshold of 1.0 to reduce the influence of very low ranked parses.

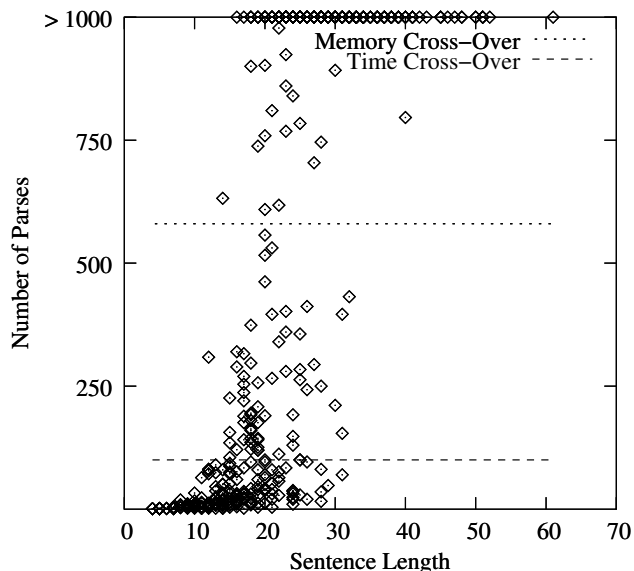


Figure 4: Scatter graph of number of parses to sentence length (one point per sentence). The cross-over points are illustrated for time and memory. The maximum number of parses shown is 1000, points plotted at 1000 correspond to equal to or greater than 1000 parses.

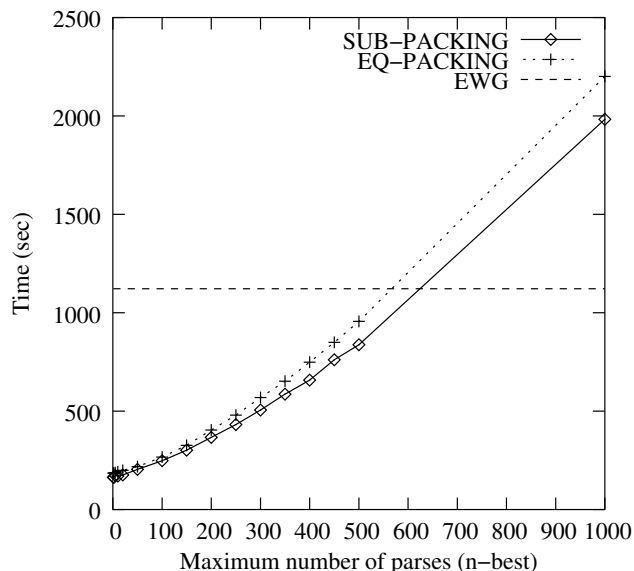


Figure 5: Comparison of total CPU time required by the different versions of the parsing system for calculation of weighted GRs over the n-best parses.

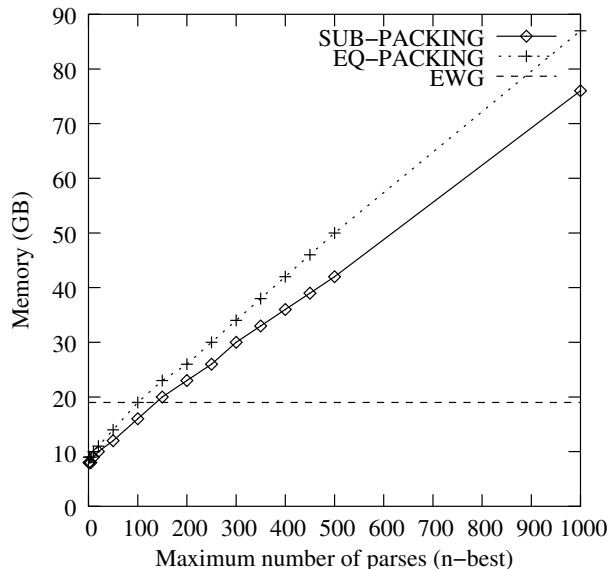


Figure 6: Comparison of total memory required by the different versions of the system for calculation of weighted GRs over the n-best parses.

a relative error reduction (RER) of 6.8%. Therefore, EWG achieves a substantial improvement in both efficiency and accuracy for weighted GR calculation; providing increased precision for thresholded GR sets and an increased  $F_1$  upper bound on the task.

## 6.2 Parse Selection

Section 6.1 illustrated the increased level of efficiency achieved by EWG compared to the current system’s method for calculating weighted GRs. This section briefly considers a parse selection algorithm using EWG that would otherwise be too inefficient to apply.

Clark and Curran (2004) determine weighted GRs directly from a packed chart using Miyao and Tsujii’s (2002) dynamic programming algorithm. They outline a parse selection algorithm which maximises the expected recall of dependencies by selecting the n-best GR set with the highest average GR score based on the weights from the weighted GRs. We can apply this parse selection algorithm in two ways: either (a) re-rank the n-best GR sets based on the average weight of GRs and select the highest ranking set, or (b) apply a simple variant of the Viterbi algorithm to select the GR set with the highest average

weighted score over the data structure built during EWG. The latter approach, based on the parse selection algorithm in Clark and Curran (2004), takes into account *all possible parses* and effectively re-ranks all parses using weights output by EWG. These approaches will be referred to as *RE-RANK* (over the top 1000 parses) and *BEST-AVG*, respectively.

The GR set corresponding to the system’s top parse achieves an  $F_1$  of 71.24%. By applying BEST-AVG and RE-RANK parse selection, we achieve a relative error reduction of 3.01% and 0.90%, respectively. Therefore, BEST-AVG achieves higher accuracy and is more efficient than RE-RANK. It is also worth noting that these parse selection schemes are able to output a *consistent* set of GRs unlike the set corresponding to high precision GR output.

## 7 Conclusions

We have described a dynamic programming approach based on the Inside Outside Algorithm for producing weighted grammatical relation output directly from a unification-based parse forest. In an evaluation on a standard test suite the approach achieves substantial improvements in accuracy and parser throughput over a previous implementation. The approach is novel as it allows multiple heads (and inside probabilities) per parse forest node instead of manipulating the parse forest so that each node represents only a single head.

We intend to extend this work to develop more sophisticated parse selection schemes based on weighted GR output. Re-ranking the n-best GR sets results in a consistent but not necessarily a coherent set of GRs. Given the increased upper bound on precision for the high precision GR output, we hope to boost the corresponding recall measure by determining a consistent and coherent set of GRs *from the weighted GR set*.

## Acknowledgements

This work is in part funded by the Overseas Research Students Awards Scheme and the Poynton Scholarship appointed by the Cambridge Australia Trust in collaboration with the Cambridge Commonwealth Trust. We would like to thank four anonymous reviewers who provided many useful suggestions for improvement.

## References

- J. K. Baker. 1979. Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America*, pages 557–550.
- Ted Briscoe and John Carroll. 1995. Developing and evaluating a probabilistic LR parser of part-of-speech and punctuation labels. In *Proceedings of the ACL/SIGPARSE 4th International Workshop on Parsing Technologies*, pages 48–58, Prague / Karlovy Vary, Czech Republic.
- Ted Briscoe and John Carroll. 2002. Robust accurate statistical annotation of general text. In *Proceedings of the Conference on Language Resources and Evaluation (LREC 2002)*, pages 1499–1504, Palmas, Canary Islands, May.
- Ted Briscoe and John Carroll. 2005. Evaluating the speed and accuracy of an unlexicalized statistical parser on the PARC Depbank. Under review.
- John Carroll and Ted Briscoe. 2002. High precision extraction of grammatical relations. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.
- John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 447–454, Granada.
- John Carroll. 1993. *Practical unification-based parsing of natural language*. Ph.D. thesis, Computer Laboratory, University of Cambridge. Technical Report No. 314.
- Stephen Clark and James Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Barcelona, Spain.
- Ann Copestake. 2003. Report on the design of RMRS. DeepThought Project Deliverable D1.1a, University of Cambridge, UK.
- Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA.
- Mark Johnson. 2001. Joint and conditional estimation of tagging and parsing models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, Toulouse, France, July.
- Ronald Kaplan, Stephen Riezler, Tracy King, John Maxwell, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology conference / North American chapter of the Association for Computational Linguistics annual meeting*, pages 97–113, Boston, Massachusetts, May.
- Tracy King, Richard Crouch, Stephen Riezler, Mary Dalrymple, and Ronald Kaplan. 2003. The PARC700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*.
- Karim Lari and Steve Young. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 2(4):35–56.
- DeKang Lin. 1998. Dependency-based evaluation of MINIPAR. In *Proceedings of the Workshop on The Evaluation of Parsing Systems at the 1st International Conference on Language Resources and Evaluation*, Granada, Spain.
- Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*, San Diego, California, March.
- Stephan Oepen and John Carroll. 2000. Ambiguity packing in constraint-based parsing - practical results. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 162–169, Seattle, WA.
- Geoffrey Sampson. 1995. *English for the Computer*. Oxford University Press.
- Helmut Schmid and Mats Rooth. 2001. Parse forest computation of expected governors. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 458–465.
- Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.