Parse Forest Computation of Expected Governors

Helmut Schmid
Institute for Computational Linguistics
University of Stuttgart
Azenbergstr. 12
70174 Stuttgart, Germany
schmid@ims.uni-stuttgart.de

Mats Rooth
Department of Linguistics
Cornell University
Morrill Hall
Ithaca, NY 14853, USA
mats@cs.cornell.edu

Abstract

In a headed tree, each terminal word can be uniquely labeled with a governing word and grammatical relation. This labeling is a summary of a syntactic analysis which eliminates detail, reflects aspects of semantics, and for some grammatical relations (such as subject of finite verb) is nearly uncontroversial. We define a notion of expected governor markup, which sums vectors indexed by governors and scaled by probabilistic tree weights. The quantity is computed in a parse forest representation of the set of tree analyses for a given sentence, using vector sums and scaling by inside probability and flow.

1 Introduction

A labeled headed tree is one in which each non-terminal vertex has a distinguished head child, and in the usual way non-terminal nodes are labeled with non-terminal symbols (syntactic categories such as NP) and terminal vertices are labeled with terminal symbols (words such as *reads*).[1] We work with syntactic trees in which terminals are in addition labeled with uninflected word forms (lemmas) derived from the lexicon. By percolating lemmas up the chains of heads, each node in a headed tree may be labeled with a lexical head. Figure 1 is an example, where lexical heads are written as subscripts. We use the notation $h(v)$ for the lexical head of a vertex v, and $c(v)$ for the ordinary category or word label of v.

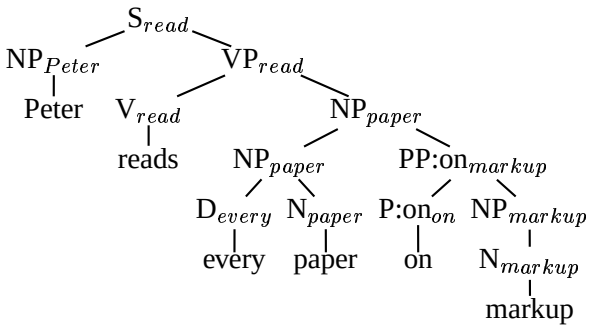

Figure 1: A tree with percolated lexical heads.

The *governor label* for a terminal vertex v in such a labeled tree is a triple which represents the syntactic and lexical environment at the top of the chain of vertices headed by v. Where u is the maximal vertex of which v is a head vertex, and u' is the parent of u, the governor label for v

The governor algorithm was designed and implemented in the Reading Comprehension research group in the 2000 Workshop on Language Engineering at Johns Hopkins University. Thanks to Marc Light, Ellen Riloff, Pranav Anand, Brianne Brown, Eric Breck, Gideon Mann, and Mike Thelen for discussion and assistance. Oral presentations were made at that workshop in August 2000, and at the University of Sussex in January 2001. Thanks to Fred Jelinek, John Carroll, and other members of the audiences for their comments.

[1] Headed trees may be constructed as tree domains, which are sets of addresses of vertices. 0 is used as the relative address of the head vertex, negative integers are used as relative addresses of child vertices before the head, and positive integers are used as relative addresses of child vertices after the head. A headed tree domain is a set of finite sequences of integers t such that (i) if $\alpha\beta\epsilon t$, then $\alpha\epsilon t$; (ii) if $\alpha n\epsilon t$ and $0 \leq k < n$ or $n < k \leq 0$, then $\alpha k\epsilon t$.

position	*word*	*governor label*
1	Peter	⟨NP,S,*read*⟩
2	reads	⟨S,STARTC,*startw*⟩
3	every	⟨D,NP,*paper*⟩
4	paper	⟨NP,VP,*read*⟩
5	on	⟨P:ON,PP:ON,*markup*⟩
6	markup	⟨NP,PP:ON,*paper*⟩

Figure 2: Governor labels for the terminals in the tree of Figure 1. For the head of the sentence, special symbols startc and startw are used as the parent category and parent lexical governor.

is the tuple $\langle c(u), c(u'), h(u')\rangle$.[2] Governor labels for the example tree are given in Figure 2.

As observed in Chomsky (1965), grammatical relations such as subject and object may be reconstructed as ordered pairs of category labels, such as ⟨NP,S⟩ for subject. So, a governor label encodes a grammatical relation and a governing lexical head.

Given a unique tree structure for a sentence, governor markup may be read off the tree. However, in view of the fact that robust broad coverage parsers frequently deliver thousands, millions, or thousands of millions of analyses for sentences of free text, basing annotation on a unique tree (such as the most probable tree analysis generated by a probabilistic grammar) appears arbitrary.

Note that different trees may produce the same governor labels for a given terminal position. Suppose for instance that the yield of the tree in Figure 1 has a different tree analysis in which the PP is a child of the VP, rather than NP. In this case, just as in the original tree, the label for the fourth terminal position (with word label *paper*) is ⟨NP,VP,*read*⟩. Supposing that there are only two tree analyses, this label can be assigned to the fourth word with certainty, in the face of syntactic ambiguity. The algorithm we will define pools governor labels in this way.

2 Expected Governors

Suppose that a probabilistic grammar licenses headed tree analyses $t_1, ..., t_n$ for a sentence σ, and assigns them probabilistic weights $p_1, ..., p_n$.

[2] In a headed tree domain, v is a head of u if v is of the form $u0^n$ for $n \geq 1$.

that	NP SC deprive	.95	.99
would	MD2 VFP deprive	.98	1
deprive	S STARTC startw	1	1
all	DETPL2 NC student	.83	.98
beginning	NSG1 NPL1 student	.75	.98
students	NP VFP deprive	.82	.98
	NP VGP begin	.16	
of	PP NP student	.53	
	PP VFP deprive	.38	.99
their	DETPL2 NC lunch	.92	.98
high	ADJMOD NPL2 lunch	.78	.23
	ADJMOD NSG2 school	.15	.76
school	NCHAIN NPL1 lunch	.16	
	NSG1 NPL1 lunch	.76	.98
lunches	NP PP of	.91	.98
.	PERC S deprive	.88	.86
	PERC X deprive		.14

Figure 3: Expected governors in the sentence *That would deprive all beginning students of their high school lunches.* For a label m in column 2, column 3 gives $E(m)$ as computed with a PCFG weighting of trees, and column 4 gives $E(m)$ as computed with a head-lexicalized weighting of trees. Values below 0.1 are omitted. According to the lexicalized model, the PP headed by *of* probably attaches to VFP (finite verb phrase) rather than NP.

Let $m_1, ..., m_n$ be the governor labels for word position k determined by $t_1, ..., t_n$ respectively. We define a scheme which divides a count of 1 among the different governor labels.

For a given governor tuple m, let

$$E(m) \stackrel{def}{=} \frac{\sum_{m_i=m} p_i}{\sum_{1\leq i\leq n} p_i}. \qquad (1)$$

The definition sums the probabilistic weights of trees with markup m, and normalizes by the sum of the probabilities of all tree analyses of σ.

The definition may be justified as follows. We work with a markup space $M = C \times C \times W$, where C is the set of category labels and W is the set of lemma labels. For a given markup triple m, let

$$b_m : M \to I\!R$$

be the function which maps m to 1, and m' to 0 for $m' \neq m$. We define a random variate

$$r : \text{Trees} \to [M \to I\!R]$$

which maps a tree t to b_m, where m is the governor markup for word position k which is de-

termined by tree t. The random variate r is defined on labeled trees licensed by the probabilistic grammar. Note that $[M \rightarrow I\!R]$ is a vector space (with pointwise sums and scalar products), so that expectations and conditional expectations may be defined. In these terms, E is the conditional expectation of r, conditioned on the yield being σ.

This definition, instead of a single governor label for a given word position, gives us a set of pairs of a markup m and a real number $E(m)$ in [0,1], such that the real numbers in the pairs sum to 1. In our implementation (which is based on Schmid (2000a)), we use a cutoff of 0.1, and print only indices m where $E(m)$ is above the cutoff. Figure 3 is an example.

A direct implementation of the above definition using an iteration over trees to compute E would be unusable because in the robust grammar of English we work with, the number of tree analyses for a sentence is frequently large, greater than 10^9 for about 1/10 of the sentences in the British National Corpus. We instead calculate E in a parse forest representation of a set of tree analyses.

3 Parse Forests

A parse forest (see also Billot and Lang (1989)) in labeled grammar notation is a tuple $P = \langle N_P, \Sigma_P, R_P, S_P, L_P \rangle$ where $\langle N_P, \Sigma_P, R_P, S_P \rangle$ is a context free grammar (consisting of non-terminals N_P, terminals Σ_P, rules R_P, and a start symbol S_P) and L_P is a function which maps elements of N_P to non-terminals in an underlying grammar $G = \langle N, \Sigma, R, S \rangle$ and elements of Σ_P to terminals in G. By using L_P on symbols on the left hand and right hand sides of a parse forest rule, L_P can be extended to map the set of parse forest rules R_P to the set of underlying grammar rules R. L_P is also extended to map trees licensed by the parse forest grammar to trees licensed by the underlying grammar. An example is given in figure 4.

Where $x \epsilon N_P \cup \Sigma_P \cup R_P$, let $I_P(x)$ be the set of trees licensed by $\langle N_P, \Sigma_P, R_P, S_P \rangle$ which have root symbol x in the case of a symbol, and the set of trees which have x as the rule expanding the root in the case or a rule. $I(x)$ is defined to be the multiset image of $I_P(x)$ under L_P. $I(x)$ is the multiset of inside trees represented by parse forest symbol or rule x.[3] Let $C_P(x)$ be the set of trees in $I_P(S_P)$ which contain x as a symbol or use x as a rule. $C(x)$ is defined to be the multiset image of $C_P(x)$ under L_P. $C(x)$ is the multiset of complete trees represented by the parse forest symbol or rule x.

$S_1 \rightarrow NP_1\ VP_1$	$NP_4 \rightarrow N_2$
$VP_1 \rightarrow V_1\ NP_2$	$NP_1 \rightarrow$ Peter
$VP_1 \rightarrow VP_2\ PP_1$	$V_1 \rightarrow$ reads
$NP_2 \rightarrow NP_3\ PP_1$	$D_1 \rightarrow$ every
$NP_3 \rightarrow D_1\ N_1$	$N_1 \rightarrow$ paper
$PP_1 \rightarrow P_1\ NP_4$	$P_1 \rightarrow$ on
$VP_2 \rightarrow V_1\ NP_3$	$N_2 \rightarrow$ markup

Figure 4: Rule set R_P of a labeled grammar representing two tree analyses of *John reads every paper on markup*. The labeling function drops subscripts, so that $L_P(\mathrm{VP}_1) = \mathrm{VP}$.

Where p is a probability function on trees licensed by the underlying grammar and x is a symbol or rule in P,

$$i(x) \stackrel{def}{=} \sum_{t \epsilon I(x)} p(t) \tag{2}$$

$$\phi(x) \stackrel{def}{=} \frac{\sum_{t \epsilon C(x)} p(t)}{\sum_{t \epsilon C(S_P)} p(t)}. \tag{3}$$

$i(x)$ is called the inside probability for x and $\phi(x)$ is called the flow for x.[4]

Parse forests are often constructed so that all inside trees represented by a parse forest nonterminal $n \epsilon N_P$ have the same span, as well as the same parent category. To deal with headedness and lexicalization of a probabilistic grammar, we construct parse forests so that, in addition, all inside trees represented by a parse forest nonterminal have the same lexical head. We add to the labeled grammar a function H_P which labels parse forest symbols with lexical heads. In our implementation, an ordinary context free parse forest is

[3] We use multisets rather than set images to achieve correctness of the inside algorithm in cases where P represents some tree more than once, something which is possible given the definition of labeled grammars. A correct parser produces a parse forest which represents every parse for the input sentence exactly once.

[4] These quantities can be given probabilistic interpretations and/or definitions, for instance with reference to conditionally expected rule frequencies for flow.

PF-INSIDE(P, w)

1 Initial. float array $i[R_P \cup N_P \cup \Sigma_P] \leftarrow 0$
2 **for** $v \in \Sigma_P$
3 **do** $i[v] \leftarrow 1$
4 **for** r in R_P in bottom-up order
5 **do** $i[r] \leftarrow w(L_P(r)) \prod_{v \in rhs(r)} i[v]$
6 $i[lhs(r)] \leftarrow i[lhs(r)] + i[r]$
7 **return** i

Figure 5: Inside algorithm.

PF-FLOW(P, i)

1 Initial. float array $\phi[R_P \cup N_P \cup \Sigma_P] \leftarrow 0$
2 $\phi[S_P] \leftarrow 1$
3 **for** r in R_P in top-down order
4 **do** $\phi[r] \leftarrow \frac{i[r]}{i[lhs(r)]} \phi[lhs(r)]$
5 **for** v in $rhs(r)$
6 **do** $\phi[v] \leftarrow \phi[v] + \phi[r]$
7 **return** ϕ

Figure 6: Flow algorithm.

first constructed by tabular parsing, and then in a second pass parse forest symbols are split according to headedness. Such an algorithm is shown in appendix B. This procedure gives worst case time and space complexity which is proportional to the fifth power of the length of the sentence. See Eisner and Satta (1999) for discussion and an algorithm with time and space requirements proportional to the fourth power of the length of the input sentence in the worst case. In practical experience with broad-coverage context free grammars of several languages, we have not observed super-cubic average time or space requirements for our implementation. We believe this is because, for our grammars and corpora, there is limited ambiguity in the position of the head within a given category-span combination.

The governor algorithm stated in the next section refers to headedness in parse forest rules. This can be represented by constructing parse forest rules (as well as ordinary grammar rules) with headed tree domains of depth one.[5] Where u is a parse forest symbol on the right hand side of a parse forest rule r, we will simply state the condition "u is the head of r".

The flow and governor algorithms stated below call an algorithm PF-INSIDE(P, w) which computes inside probabilities in P, where w is a function giving probability parameters for the underlying grammar. Any probability weighting of trees may be used which allows inside probabilities to be computed in parse forests. The inside algorithm for ordinary PCFGs is given in figure 5. The parameter w maps the set of underlying grammar rules R which is the image of L_P on L_R to reals, with the interpretation of rule probabilities. In step 5, L_P maps the parse forest rule r to a grammar rule $L_P(r)$ which is the argument of w. The functions *lhs* and *rhs* map rules to their left hand and right hand sides, respectively.

Given an inside algorithm, the flow ϕ may be computed by the flow algorithm in Figure 6, or by the inside-outside algorithm.

4 Governors Algorithm

The governor algorithm annotates parse forest symbols and rules with functions from governor labels to real numbers. Let t be a tree in the parse forest grammar, let v be a symbol in t, let u be the maximal symbol in t of which v is a head, or v itself if v is a non-head child of its parent in t, and let u' be the parent of u in t. Recall that

$$b_{L_P(u), L_P(u'), H_P(u')} \tag{4}$$

is a vector mapping the markup triple $\langle L_P(u), L_P(u'), H_P(u') \rangle$ to 1 and other markups to 0. We have constructed parse forests such that $\langle L_P(u), L_P(u'), H_P(u') \rangle$ agrees with the governor label for the lexical head of the node corresponding to v in $L_P(t)$.

A parse forest tree t and symbol v in t thus determine the vector (4), where u and u' are defined as above. Call the vector determined in this way $b(t, v)$. Where v is parse forest symbol in P and r is a parse forest rule in P, let

$$\gamma(v) \stackrel{def}{=} \frac{\sum_{t \epsilon C_P(v)} p(L_P(t)) b(t, v)}{\sum_{t \epsilon C_P(S_P)} p(L_P(t))} \tag{5}$$

[5] See footnote 1. Constructed in this way, the first rule in parse forest in Figure 4 has domain $\{\epsilon, -1, 0\}$, and labeling function $\{\langle \epsilon, \mathrm{S}_1 \rangle, \langle -1, \mathrm{NP}_1 \rangle, \langle 0, \mathrm{VP}_1 \rangle\}$. When parse forest rules are mapped to underlying grammar rules, the domain is preserved, so that L_P applied to the parse forest rule just described is the tree with domain $\{\epsilon, -1, 0\}$ and label function $\{\langle \epsilon, \mathrm{S} \rangle, \langle -1, \mathrm{NP} \rangle, \langle 0, \mathrm{VP} \rangle\}$. ϵ is the empty string.

PF-GOVERNORS(P, w)
1 $i \leftarrow$ PF-INSIDE(P, w)
2 $\phi \leftarrow$ PF-FLOW(P, i)
3 Initialize array $\gamma[R_P \cup N_P \cup \Sigma_P]$ to empty maps from governor labels to float
4 $\gamma[S_P] \leftarrow b_{C_P(S_P),\text{startc},\text{startw}}$
5 **for** r in R_P in top-down order
6 **do** $\gamma[r] \leftarrow \frac{i[r]}{i[lhs(r)]}\gamma[lhs(r)]$
7 **for** u in $rhs(r)$
8 **do if** u is the head of r
9 **then** $\gamma[u] \leftarrow \gamma[u] + \gamma[r]$
10 **else** $\gamma[u] \leftarrow \gamma[u] + \phi[r]b_{C_P(u),C_P(lhs(r)),H_P(lhs(r))}$
11 **return** γ

Figure 7: Parse forest computation of governor vector.

$$\gamma(r) \stackrel{def}{=} \frac{\sum_{t \epsilon C_P(r)} p(L_P(t)) b(t, lhs(r))}{\sum_{t \epsilon C_P(S_P)} p(L_P(t))}. \quad (6)$$

Assuming that $P = \langle N_P, \Sigma_P, R_P, S_P, L_P \rangle$ is a parse forest representing each tree analysis for a sentence exactly once, the quantity E for terminal position k (as defined in section 1) is found by summing $\gamma(v)$ for terminal symbols v in Σ_P which have string position k.[6]

The algorithm PF-GOVERNORS is stated in Figure 3. Working top down, if fills in an array $\gamma[\cdot]$ which is supposed to agree with the quantity $\gamma(\cdot)$ defined above. Scaled governor vectors are created for non-head children in step 10, and summed down the chain of heads in step 9. In step 6, vectors are divided in proportion to inside probabilities (just as in the flow algorithm), because the set of complete trees for the left hand side of r are partitioned among the parse forest rules which expand the left hand side of r.

Consider a parse forest rule r, and a parse forest symbol u on its right hand side which is not the head of r. In each tree in $C_P(r)$, u is the top of a chain of heads, because u is a non-head child in rule r. In step 10, the governor tuple describing the syntactic environment of u in trees in $C_P(r)$ (or rather, their images under L_P) is constructed as $b_{C_P(u),C_P(lhs(r)),H_P(lhs(r))}$. The scalar multiplier $\phi[r]$ is

$$\frac{\sum_{t \epsilon C_P(r)} p(t)}{\sum_{t \epsilon C_P(S_P)} p(t)},$$

the relative weight of trees in $C_P(r)$. This is appropriate because $\gamma(u)$ as defined in equation (5) is to be scaled by the relative weight of trees in $C_P(u)$.

In line 9 of the algorithm, γ is summed into the head child u. There is no scaling, because every tree in $C_P(r)$ is a tree in $C_P(u)$.

A probability parameter vector w is used in the inside algorithm. In our implementation, we can use either a probabilistic context free grammar, or a lexicalized context free grammar which conditions rules on parent category and parent lexical head, and conditions the heads of non-head children on child category, parent category, and parent head (Eisner, 1997; Charniak, 1995; Carroll and Rooth, 1998). The requisite information is directly represented in our parse forests by C_P and H_P. Thus the call to PF-INSIDE in line 1 of PF-GOVERNORS may involve either a computation of PCFG inside probabilities, or head-lexicalized inside probabilities. However, in both cases the algorithm requires that the parse forest symbols be split according to heads, because of the reference to H_P in line 10. Construction of head-marked parse forests is presented in the appendix.

The LoPar parser (Schmid, 2000a) on which our implementation of the governor algorithm is

[6] This procedure requires that symbols in Σ_P correspond to a unique string position, something which is not enforced by our definition of parse forests. Indeed, such cases may arise if parse forest symbols are constructed as pairs of grammar symbols and strings (Tendeau, 1998) rather than pairs of grammar symbols and spans. Our parser constructs parse forests organized according to span.

based represents the parse forest as a graph with at most binary branching structure. Nodes with more than two daughter nodes in a conventional parse forest are replaced with a right-branching tree structure and common sub-trees are shared between different analyses. The worst-case space complexity of this representation is cubic (cmp. Billot and Lang (1989)).

LoPar already provided functions for the computation of the head-marked parse forest, for the flow computation and for traversing the parse forest in depth-first and topologically-sorted order (see Cormen et al. (1994)). So it was only necessary to add functions for data initialization, for the computation of the governor vector at each node and for printing the result.

5 Pooling of grammatical relations

The governor labels defined above are derived from the specific symbols of a context free grammar. In contrast, according to the general markup methodology of current computational linguistics, labels should not be tied to a specific grammar and formalism. The same markup labels should be produced by different systems, making it possible to substitute one system for another, and to compare systems using objective tests.

Carroll et al. (1998) and Carroll et al. (1999) propose a system of grammatical relation markup to which we would like to assimilate our proposal. As grammatical relation symbols, they use atomic labels such as *dobj* (direct object) an *ncsubj* (non-clausal subject). The labels are arranged in a hierarchy, with for instance *subj* having subtypes *ncsubj*, *xsubj*, and *csubj*.

There is another problem with the labels we have used so far. Our grammar codes a variety of features, such as the feature VFORM on verb projections. As a result, instead of a single object grammatical relation ⟨NP,VP⟩, we have grammatical relations ⟨NP,VP.N⟩, ⟨NP,VP.FIN⟩, ⟨NP,VP.TO⟩, ⟨NP,VP.BASE⟩, and so forth. This may result in frequency mass being split among different but similar labels. For instance, a verb phrase *will have read every paper* might have some analyses in which *read* is the head of a base form VP and *paper* is the head of the object of *read*, and others where *read* is a head of a finite form VP, and *paper* is the head of the object of *read*. In this case, frequencies would be split between ⟨NP,VP.BASE,*read*⟩ and ⟨NP,VP.FIN,*read*⟩ as governor labels for *paper*.

To address these problems, we employ a pooling function GR which maps pairs of categories to symbols such as *ncsubj* or *obj*. The governor tuple $\langle c(u), c(u'), h(u')\rangle$ is then replaced by $\langle GR(c(u), c(u')), h(u)\rangle$ in the definition of the governor label for a terminal vertex v. Line 10 of PF-GOVERNORS is changed to

$$\gamma[u] \leftarrow \gamma[u] + \phi[r] b_{GR(C_P(u), C_P(lhs(r))), H_P(lhs(r))}.$$

More flexibility could be gained by using a rule and the address of a constituent on the right hand side as arguments of GR. This would allow the following assignments.

$$GR(\text{VP.FIN} \rightarrow \text{VC.FIN' NP NP}, 1) = \textit{dobj}$$
$$GR(\text{VP.FIN} \rightarrow \text{VC.FIN' NP NP}, 2) = \textit{obj2}$$
$$GR(\text{VP.FIN} \rightarrow \text{VC.FIN' VP.TO}, 1) = \textit{xcomp}$$
$$GR(\text{VP.FIN} \rightarrow \text{VP.FIN' VP.TO}, 1) = \textit{xmod}$$

The head of a rule is marked with a prime. In the first pair, the objects in double object construction are distinguished using the address. In each case, the child-parent category pair is ⟨NP,VP.FIN⟩, so that the original proposal could not distinguish the grammatical relations. In the second pair, a VP.TO argument is distinguished from a VP.TO modifier using the category of the head. In each case, the child-parent category pair is ⟨VP.TO,VP.FIN⟩. Notice that in Line 10 of PF-GOVERNORS, the rule r is available, so that the arguments of GR could be changed in this way.

6 Discussion

The governor algorithm was designed as a component of Spot, a free-text question answering system. Current systems usually extract a set of candidate answers (e.g. sentences), score them and return the n highest-scoring candidates as possible answers. The system described in Harabagiu et al. (2000) scores possible answers based on the overlap in the semantic representations of the question and the answer candidates. Their semantic representation is basically identical to the head-head relations computed by the governor algorithm. However, Harabagiu

et al. extract this information only from maximal probability parses whereas the governor algorithm considers all analyses of a sentence and returns all possible relations weighted with estimated frequencies. Our application in Spot works as follows: the question is parsed with a specialized question grammar, and features including the governor of the trace are extracted from the question. Governors are among the features used for ranking sentences, and answer terms within sentences. In collaboration with Pranav Anand and Eric Breck, we have incorporated governor markup in the question answering prototype, but not debugged or evaluated it.

Expected governor markup summarizes syntactic structure in a weighted parse forest which is the product of exhaustive parsing and inside-outside computation. This is a strategy of dumbing down the product of computationally intensive statistical parsing into unstructured markup. Estimated frequency computations in parse forests have previously been applied to tagging and chunking (Schulte im Walde and Schmid, 2000). Governor markup differs in that it is reflective of higher-level syntax. The strategy has the advantage, in our view, that it allows one to base markup algorithms on relatively sophisticated grammars, and to take advantage of the lexically sensitive probabilistic weighting of trees which is provided by a lexicalized probability model.

Localizing markup on the governed word increases pooling of frequencies, because the span of the phrase headed by the governed item is ignored. This idea could be exploited in other markup tasks. In a chunking task, categories and heads of chunks could be identified, rather than categories and boundaries.

References

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the ACL, University of British Columbia*, Vancouver, B.C., Canada.

Glenn Carroll and Mats Rooth. 1998. Valence induction with a head-lexicalized PCFG. In *Proceedings of Third Conference on Empirical Methods in Natural Language Processing*, Granada, Spain.

John Carroll, Antonio Sanfilippo, and Ted Briscoe. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the International Conference of Language Resources and Evaluation*, pages 447–454, Granada, Spain.

John Carroll, Guido Minnen, and Ted Briscoe. 1999. Corpus annotation for parser evaluation. In *Proceedings of the EACL99 workshop on Linguistically Interpreted Corpora (LINC)*, Bergen, Norway, June.

Eugene Charniak. 1993. *Statistical Language Learning*. The MIT Press, Cambridge, Massachusetts.

Eugene Charniak. 1995. Parsing with context-free grammars and word statistics. Technical Report CS-95-28, Department of Computer Science, Brown University.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. M.I.T. Press, Cambridge, MA.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1994. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, College Park, MD.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 4th international Workshop on Parsing Technologies*, Cambridge, MA.

S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Gîrju, V. Rus, and P. Morarescu. 2000. Falcon: Boosting knowledge for answer engines. In *Proceedings of the Ninth Text REtrieval Conference (TREC 9)*, Gaithersburg, MD, USA, November.

Helmut Schmid. 2000a. *LoPar: Design and Implementation*. Number 149 in Arbeitspapiere des Sonderforschungsbereiches 340. Institute for Computational Linguistics, University of Stuttgart.

Helmut Schmid. 2000b. Lopar man pages. Institute for Computational Linguistics, University of Stuttgart.

Sabine Schulte im Walde and Helmut Schmid. 2000. Robust german noun chunking with a probabilistic context-free grammar. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 726–732, Saarbrücken, Germany, August.

Frederic Tendeau. 1998. Computing abstract decorations of parse forests using dynamic programming and algebraic power series. *Theoretical Computer Science*, (199):145–166.

A Relation Between Flow and Inside-Outside Algorithm

The inside-outside algorithm computes inside probabilities $i[v]$ and outside probabilities $o[v]$. We will show that these quantities are related to the flow $\phi(v)$ by the equation $\phi[v] = o[v]i[v]/i[S_P]$. $i[S_P]$ is the inside probability of the root symbol, which is also the sum of the probabilities of all parse trees.

According to Charniak (1993), the outside probabilities in a parse forest are computed by:

$$o[v] = \sum_{r:\, v \in rhs(r)} o[lhs(r)]\ \frac{i[r]}{i[v]}$$

The outside probability of the start symbol is 1. We prove by induction over the depth of the parse forest that the following relationship holds:

$$\phi[v] = o[v]\ \frac{i[v]}{i[S_P]}$$

It is easy to see that the assumption holds for the root symbol S_P:

$$\phi[S_P] = 1 = o[S_P]\ \frac{i[S_P]}{i[S_P]}$$

The flow in a parse forest is computed by:

$$\phi[v] = \sum_{r:\, v \in rhs(r)} \phi[lhs(r)] \frac{i[r]}{i[lhs(r)]}$$

Now, we insert the induction hypothesis:

$$\phi[v] = \sum_{r:\, v \in rhs(r)} \frac{o[lhs(r)]\ i[lhs(r)]\ i[r]}{i[S_P]\ i[lhs(r)]}$$

After a few transformations, we get the equation

$$\phi[v] = \frac{i[v]}{i[S_P]} \sum_{r:\, v \in rhs(r)} \frac{o[lhs(r)] i[r]}{i[v]}$$

which is equivalent to

$$\phi[v] = o[v] \frac{i[v]}{i[S_P]}$$

according to the definition of $o[v]$. So, the induction hypothesis is generally true.

B Parse Forest Lexicalization

The function LEXICALIZE below takes an unlexicalized parse forest as argument and returns a lexicalized parse forests, where each symbol is uniquely labeled with a lexical head. Symbols are split if they have more than one lexical head.

```
LEXICALIZE(P)
 1  initialize P' as an empty parse forest
 2  initialize array l[N_P ∪ Σ_P] ← ∅
 3  for v in Σ_P
 4       do v' ← NEWT(word(v))
 5          l[v] ← {v'}
 6  for r in R_P in bottom-up order
 7       do assume rhs(r) = {v_1, v_2, ..., v_n}
 8          assume v_i is the head of r
 9          for v'_1 v'_2 ... v'_n ∈ l[v_1] × ... × l[v_n]
10               do if v_i ∈ Σ_P
11                    then h ← LEM(L_P(r))
12                    else   h ← H_P[v'_i]
13                  d' ← ⟨v'_1 ... v'_n⟩
14                  v_m ← lhs(r)
15                  r' ←  ADD(P', r, h, d')
16                  l[v_m] ← l[v_m] ∪ {lhs(r')}
17  return P'
```

LEXICALIZE creates new terminal symbols by calling the function NEWT. The new symbols are linked to the original ones by means of $l[\cdot]$. For each rule in the old parse forest, the set of all possible combinations of the lexicalized daughter symbols is generated. The function LEM(r) returns the lemma associated with lexical rule r.

```
ADD(P, r, h, d)
 1  if ∃v ∈ l[lhs(r)] s.t. H_P[v] = h
 2     then v' ← v
 3     else   v' ←NEWNT(P)
 4            L_P(v') ← L_P(lhs(r))
 5            H_P[v'] ← h
 6  r' ←NEWRULE(P, v', d)
 7  L_P(r') ← L_P(r)
 8  return r'
```

For each combination of lexicalized daughter symbols, a new rule is inserted by calling ADD. ADD calls NEWNT to create new non-terminals and NEWRULE to generate new rules. A non-terminal is only created if no symbol with the same lexical head was linked to the original node.