A Statistical Theory of Dependency Syntax

Christer Samuelsson
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan, FRANCE
Christer.Samuelsson@xrce.xerox.com

Abstract

A generative statistical model of dependency syntax is proposed based on Tesnière's classical theory. It provides a stochastic formalization of the original model of syntactic structure and augments it with a model of the string realization process, the latter which is lacking in Tesnière's original work. The resulting theory models crossing dependency links, discontinuous nuclei and string merging, and it has been given an efficient computational rendering.

1 Introduction

The theory of dependency grammar culminated in the seminal book by Lucien Tesnière, (Tesnière, 1959), to which also today's leading scholars pay homage, see, e.g., (Mel'čuk, 1987). Unfortunately, Tesnière's book is only available in French, with a partial translation into German, and subsequent descriptions of his work reported in English, (Hays, 1964), (Gaifman, 1965), (Robinson, 1970), etc., stray increasingly further from the original, see (Engel, 1996) or (Järvinen, 1998) for an account of this.

The first step when assigning a dependency description to an input string is to segment the input string into nuclei. A nucleus can be a word, a part of a word, or a sequence of words and subwords, and these need not appear contiguously in the input string. The best way to visualize this is perhaps the following: the string is tokenized into a sequence of tokens and each nucleus consists of a subsequence of these tokens. Alternative readings may imply different ways of dividing the token sequence into nuclei, and segmenting the input string into nuclei is therefore in general a nondeterministic process.

The next step is to relate the nuclei to each other through dependency links, which are directed and typed. If there is a dependency link from one nucleus to another, the former is called a dependent of the latter, and the latter a regent of the former. Theories of dependency syntax typically require that each nucleus, save a single root nucleus, is assigned a unique regent, and that there is no chain of dependency links that constitutes a cycle. This means that the dependency links establish a tree structure,

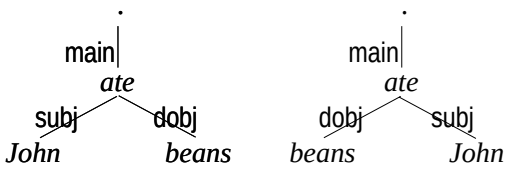

Figure 1: Dependency trees for *John ate beans*.

where each node is labeled by a nucleus. Thus, the label assigned to a node is a dependent of the label assigned to its parent node, and conversely, the label assigned to a node is the regent of the labels assigned to its child nodes. Figure 1 shows two dependency trees for the sentence *John ate beans*.

In Tesnière's dependency syntax, only the dependency structure, not the order of the dependents, is represented by a dependency tree. This means that dependency trees are unordered, and thus that the two trees of Figure 1 are equivalent. This also means that specifying the surface-string realization of a dependency description becomes a separate issue.

We will model dependency descriptions as two separate stochastic processes: one top-down process generating the tree structure $\mathcal{T}$ and one bottom-up process generating the surface string(s) $\mathcal{S}$ given the tree structure:

$$P(\mathcal{T},\mathcal{S}) \quad = \quad P(\mathcal{T}) \cdot P(\mathcal{S} \mid \mathcal{T})$$

This can be viewed as a variant of Shannon's noisy channel model, consisting of a language model of tree structures and a signal model converting trees to surface strings. In Section 2 we describe the top-down process generating tree structures and in Section 3 we propose a series of increasingly more sophisticated bottom-up processes generating surface strings, which result in grammars with increasingly greater expressive power. Section 4 describes how the proposed stochastic model of dependency syntax was realized as a probabilistic chart parser.

2 Generating Dependency Trees

To describe a tree structure $\mathcal{T}$, we will use a string notation, introduced in (Gorn, 1962), for the nodes

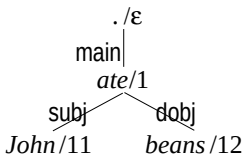

Figure 2: Gorn's tree notation for *John ate beans*.

$\mathcal{N}$	$\mathcal{L}$	$\mathcal{F}$	$\mathcal{S}$
ϵ	.	`[main]`	$s(1)$.
1	*ate*	`[subj,dobj]`	$s(11)$ *ate* $s(12)$
11	*John*	$\emptyset$	*John*
12	*beans*	$\emptyset$	*beans*

Figure 3: Dependency encoding of *John ate beans*.

of the tree, where the node name specifies the path from the root node ϵ to the node in question.

If ϕj is a node of the tree $\mathcal{T}$,
with $j \in N_+$ and $\phi \in N_+^*$,
then ϕ is also a node of the tree $\mathcal{T}$
and ϕj is a child of ϕ.

Here, N_+ denotes the set of positive integers $\{1, 2, \ldots\}$ and N_+^* is the set of strings over N_+. This means that the label assigned to node ϕj is a dependent of the label assigned to node ϕ. The first dependency tree of Figure 1 is shown in Figure 2 using this notation.

We introduce three basic random variables, which incrementally generate the tree structure:

- $\mathcal{L}(\phi) = l$ assigns the label l to node ϕ, where l is a nucleus, i.e., it is drawn from the set of strings over the set of tokens.
- $\mathcal{D}(\phi j) = d$ indicates the dependency type d linking the label of node ϕj to its regent, the label of node ϕ.
- $\mathcal{V}(\phi) = v$ indicates that node ϕ has exactly v child nodes.

Note the use of $\mathcal{V}(\phi) = 0$, rather than a partitioning of the labels into terminal and nonterminal symbols, to indicate that ϕ is a leaf node.

Let D be the (finite) set of possible dependency types. We next introduce the composite variables $\mathcal{F}(\phi)$ ranging over the power bag[1] N^D, indicating the bag of dependency types of ϕ's children:

$$\begin{aligned} &\mathcal{F}(\phi) = f = [d_1, \ldots, d_v] \quad \Leftrightarrow \\ &\quad \Leftrightarrow \quad \mathcal{V}(\phi) = v \ \wedge \ \forall_{j \in \{1,\ldots,v\}} \ \mathcal{D}(\phi j) = d_j \end{aligned}$$

Figure 3 encodes the dependency tree of Figure 2 accordingly. We will ignore the last column for now.

[1] A bag (multiset) can contain several tokens of the same type. We denote sets $\{\ldots\}$, bags $[\ldots]$ and ordered tuples $\langle\ldots\rangle$, but overload $\cup, \subseteq$, etc.

We introduce the probabilities

$$\begin{aligned} &P_{\mathcal{L}}(\epsilon) \ = \\ &\quad = \ P(\mathcal{L}(\epsilon) = l) \\ &P_{\mathcal{L}}(\phi j) \ = \\ &\quad = \ P(\mathcal{L}(\phi j) = l_j \mid \mathcal{L}(\phi) = l, \mathcal{D}(\phi j) = d_j) \\ &P_{\mathcal{F}}(\epsilon) \ = \\ &\quad = \ P(\mathcal{F}(\epsilon) = f \mid \mathcal{L}(\epsilon) = l) \\ &P_{\mathcal{F}}(\phi) \ = \ \{ \text{ for } \phi \neq \epsilon \ \} \\ &\quad = \ P(\mathcal{F}(\phi) = f \mid \mathcal{L}(\phi) = l, \mathcal{D}(\phi) = d) \end{aligned}$$

These probabilities are typically model parameters, or further decomposed into such. $P_{\mathcal{L}}(\phi j)$ is the probability of the label $\mathcal{L}(\phi j)$ of a node given the label $\mathcal{L}(\phi)$ of its regent and the dependency type $\mathcal{D}(\phi j)$ linking them. Relating $\mathcal{L}(\phi j)$ and $\mathcal{L}(\phi)$ yields lexical collocation statistics and including $\mathcal{D}(\phi j)$ makes the collocation statistics lexical-functional. $P_{\mathcal{F}}(\phi)$ is the probability of the bag of dependency types $\mathcal{F}(\phi)$ of a node given its label $\mathcal{L}(\phi)$ and its relation $\mathcal{D}(\phi)$ to its regent. This reflects the probability of the label's valency, or lexical-functional complement, and of optional adjuncts. Including $\mathcal{D}(\phi)$ makes this probability situated in taking its current role into account.

These allow us to define the tree probability

$$P(\mathcal{T}) \quad = \quad \prod_{\phi \in \mathcal{N}} P_{\mathcal{L}}(\phi) \cdot P_{\mathcal{F}}(\phi)$$

where the product is taken over the set of nodes $\mathcal{N}$ of the tree.

We generate the random variables $\mathcal{L}$ and $\mathcal{F}$ using a top-down stochastic process, where $\mathcal{L}(\phi)$ is generated before $\mathcal{F}(\phi)$. The probability of the conditioning material of $P_{\mathcal{L}}(\phi j)$ is then known from $P_{\mathcal{L}}(\phi)$ and $P_{\mathcal{F}}(\phi)$, and that of $P_{\mathcal{F}}(\phi j)$ is known from $P_{\mathcal{L}}(\phi j)$ and $P_{\mathcal{F}}(\phi)$. Figure 3 shows the process generating the dependency tree of Figure 2 by reading the $\mathcal{L}$ and $\mathcal{F}$ columns downwards in parallel, $\mathcal{L}$ before $\mathcal{F}$:

$\mathcal{L}(\epsilon) = .$, $\mathcal{F}(\epsilon) =$ `[main]`,
$\mathcal{L}(1) = ate$, $\mathcal{F}(1) =$ `[subj,dobj]`,
$\mathcal{L}(11) = John$, $\mathcal{F}(11) = \emptyset$,
$\mathcal{L}(12) = beans$, $\mathcal{F}(12) = \emptyset$

Consider calculating the probabilities at node 1:

$$\begin{aligned} &P_{\mathcal{L}}(1) \ = \\ &\quad = \ P(\mathcal{L}(1) = ate \mid \mathcal{L}(\epsilon) = ., \mathcal{D}(1) = \texttt{main}) \\ &P_{\mathcal{F}}(1) \ = \\ &\quad = \ P(\mathcal{F}(1) = [\texttt{subj}, \texttt{dobj}] \mid \\ &\qquad \mid \mathcal{L}(1) = ate, \mathcal{D}(1) = \texttt{main}) \end{aligned}$$

3 String Realization

The string realization cannot be uniquely determined from the tree structure. To model the string-realization process, we introduce another fundamental random variable $\mathcal{S}(\phi)$, which denotes the string

associated with node ϕ and which should not be confused with the node label $\mathcal{L}(\phi)$. We will introduce yet another fundamental random variable $\mathcal{M}(\phi)$ in Section 3.2, when we accommodate crossing dependency links. In Section 3.1, we present a projective stochastic dependency grammar with an expressive power not exceeding that of stochastic context-free grammars.

3.1 Projective Dependency Grammars

We let the stochastic process generating the $\mathcal{L}$ and $\mathcal{F}$ variables be as described above. We then define the stochastic string-realization process by letting the $\mathcal{S}(\phi)$ variables, given ϕ's label $l(\phi)$ and the bag of strings $s(\phi j)$ of ϕ's child nodes, randomly permute and concatenate them according to the probability distributions of the model:

$$\begin{aligned} P_{\mathcal{S}}(\epsilon) &= \\ &= P(\mathcal{S}(\epsilon) = s(\epsilon) \mid \mathcal{L}(\epsilon), \mathcal{F}(\epsilon), C(\epsilon)) \\ P_{\mathcal{S}}(\phi) &= \{ \text{ for } \phi \neq \epsilon \} \\ &= P(\mathcal{S}(\phi) = s(\phi) \mid \mathcal{D}(\phi), \mathcal{L}(\phi), \mathcal{F}(\phi), C(\phi)) \end{aligned}$$

where

$$\begin{aligned} C(\phi) &= \bigcup_{j=1}^{v} [s(\phi j)] \\ \mathcal{S}(\phi) &= \text{adjoin}(C(\phi), l(\phi)) \\ \text{adjoin}(A, \beta) &= \text{concat}(\text{permute}(A \cup [\beta])) \end{aligned}$$

The latter equations should be interpreted as defining the random variable $\mathcal{S}$, rather than specifying its probability distribution or some possible outcome. This means that each dependent is realized adjacent to its regent, where we allow intervening siblings, and that we thus stay within the expressive power of stochastic context-free grammars.

We define the string-realization probability

$$P(\mathcal{S} \mid \mathcal{T}) = \prod_{\phi \in \mathcal{N}} P_{\mathcal{S}}(\phi)$$

and the tree-string probability as

$$P(\mathcal{T}, \mathcal{S}) = P(\mathcal{T}) \cdot P(\mathcal{S} \mid \mathcal{T})$$

The stochastic process generating the tree structure is as described above. We then generate the string variables $\mathcal{S}$ using a bottom-up stochastic process. Figure 3 also shows the process realizing the surface string *John ate beans* from the dependency tree of Figure 2 by reading the $\mathcal{S}$ column upwards:

$$\mathcal{S}(12) = \textit{beans}, \ \mathcal{S}(11) = \textit{John}, \\ \mathcal{S}(1) = s(11) \ \textit{ate} \ s(12), \ \mathcal{S}(\epsilon) = s(1) \ .$$

Consider calculating the string probability at node 1. $P_{\mathcal{S}}$ is the probability of the particular permutation observed of the strings of the children and the

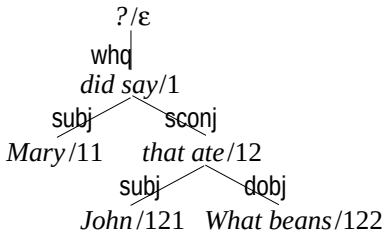

Figure 4: Dependency tree for *What beans did Mary say that John ate?*

label of the node. To overcome the sparse-data problem, we will generalize over the actual strings of the children to their dependency types. For example, $s(\texttt{subj})$ denotes the string of the subject child, regardless of what it actually might be.

$$\begin{aligned} P_{\mathcal{S}}(1) &= P(\mathcal{S}(1) = s(\texttt{subj}) \ \textit{ate} \ s(\texttt{dobj}) \mid \\ &\quad \mid \mathcal{D}(1) = \texttt{main}, \mathcal{L}(1) = \textit{ate}, \\ &\quad C(1) = [s(\texttt{subj}), s(\texttt{dobj})]) \end{aligned}$$

This is the probability of the permutation

$$\langle s(\texttt{subj}), \textit{ate}, s(\texttt{dobj}) \rangle$$

of the bag

$$[s(\texttt{subj}), \textit{ate}, s(\texttt{dobj})]$$

given this bag and the fact that we wish to form a main, declarative clause. This example highlights the relationship between the node strings and both Saussure's notion of constituency and the positional schemata of, amongst others, Didrichsen.

3.2 Crossing Dependency Links

To accommodate long-distance dependencies, we allow a dependent to be realized adjacent to the label of any node that dominates it, immediately or not. For example, consider the dependency tree of Figure 4 for the sentence *What beans did Mary say that John ate?* as encoded in Figure 5. Here, *What beans* is a dependent of *that ate*, which in turn is a dependent of *did say*, and *What beans* is realized between *did* and *say*. This phenomenon is called movement in conjunction with phrase-structure grammars. It makes the dependency grammar non-projective, since it creates crossing dependency links if the dependency trees also depict the word order.

We introduce variables $\mathcal{M}(\phi)$ that randomly select from $C(\phi)$ a subbag $C_M(\phi)$ of strings passed up to ϕ's regent:

$$\begin{aligned} C(\phi) &= \bigcup_{j=1}^{v} ([s(\phi j)] \cup C_M(\phi j)) \\ C_M(\phi) &\subseteq C(\phi) \\ P_{\mathcal{M}}(\phi) &= P(\mathcal{M}(\phi) = C_M(\phi) \mid \\ &\quad \mid \mathcal{D}(\phi), \mathcal{L}(\phi), \mathcal{F}(\phi), C(\phi)) \end{aligned}$$

$\mathcal{N}$	$\mathcal{L}$	$\mathcal{F}$
ϵ	?	`[whq]`
1	*did say*	`[subj,sconj]`
11	*Mary*	$\emptyset$
12	*that ate*	`[subj,dobj]`
121	*John*	$\emptyset$
122	*What beans*	$\emptyset$

Figure 5: Dependency encoding of *What beans did Mary say that John ate?*

$\mathcal{N}$	$\mathcal{M}$	$\mathcal{S}$
ϵ	$\emptyset$	$s(1)$?
1	$\emptyset$	$C_M(12)(s(122))$ *did* $s(11)$ *say* $s(12)$
11	$\emptyset$	*Mary*
12	$[s(122)]$	*that* $s(121)$ *ate*
121	$\emptyset$	*John*
122	$\emptyset$	*What beans*

Figure 6: Process generating *What beans did Mary say that John ate?*

The rest of the strings, $C_S(\phi)$, are realized here:

$$\begin{aligned} C_S(\phi) &= C(\phi) \setminus C_M(\phi) \\ P_{\mathcal{S}}(\phi) &= P(\mathcal{S}(\phi) = s(\phi) \mid \mathcal{D}(\phi), \mathcal{L}(\phi), \mathcal{F}(\phi), C_S(\phi)) \\ \mathcal{S}(\phi) &= \text{adjoin}(C_S(\phi), l(\phi)) \end{aligned}$$

3.3 Discontinuous Nuclei

We generalize the scheme to discontinuous nuclei by allowing $\mathcal{S}(\phi)$ to insert the strings of $C_S(\phi)$ anywhere in $l(\phi)$: [2]

$$\begin{aligned} &\text{adjoin}(A, \beta) = \\ &\quad = \text{concat}(\text{permute}_{i<j \Rightarrow b_i \prec b_j}(A \cup \bigcup_{j=1}^{m} [b_j])) \\ &\beta = b_1 \ldots b_m \end{aligned}$$

This means that strings can only be inserted into ancestor labels, not into other strings, which enforces a type of reverse island constraint. Note how in Figure 6 *John* is inserted between *that* and *ate* to form the subordinate clause *that John ate*.

We define the string-realization probability

$$P(\mathcal{S} \mid \mathcal{T}) = \prod_{\phi \in \mathcal{N}} P_{\mathcal{M}}(\phi) \cdot P_{\mathcal{S}}(\phi)$$

and again define the tree-string probability

$$P(\mathcal{T}, \mathcal{S}) = P(\mathcal{T}) \cdot P(\mathcal{S} \mid \mathcal{T})$$

[2] $x \prec y$ indicates that x precedes y in the resulting permutation. Tesnière's original implicit definition of a nucleus actually does not require that the order be preserved when realizing it; if *has eaten* is a nucleus, so is *eaten has*. This is obviously a useful feature for modeling verb chains in German subordinate clauses.

To avoid derivational ambiguity when generating a tree-string pair, i.e., have more than one derivation generate the same tree-string pair, we require that no string be realized adjacent to the string of any node it was passed up through. This introduces the practical problem of ensuring that zero probability mass is assigned to all derivations violating this constraint. Otherwise, the result will be approximating the parse probability with a derivation probability, as described in detail in (Samuelsson, 2000) based on the seminal work of (Sima'an, 1996). Schemes like (Alshawi, 1996) tacitly make this approximation.

The tree-structure variables $\mathcal{L}$ and $\mathcal{F}$ are generated just as before. We then generate the string variables $\mathcal{S}$ and $\mathcal{M}$ using a bottom-up stochastic process, where $\mathcal{M}(\phi)$ is generated before $\mathcal{S}(\phi)$. The probability of the conditioning material of $P_{\mathcal{M}}(\phi)$ is then known either from the top-down process or from $P_{\mathcal{M}}(\phi j)$ and $P_{\mathcal{S}}(\phi j)$, and that of $P_{\mathcal{S}}(\phi)$ is known either from the top-down process, or from $P_{\mathcal{M}}(\phi)$, $P_{\mathcal{M}}(\phi j)$ and $P_{\mathcal{S}}(\phi j)$. The coherence of $\mathcal{S}(\phi)$ and $\mathcal{M}(\phi)$ is enforced by explicit conditioning.

Figure 5 shows a top-down process generating the dependency tree of Figure 4; the columns $\mathcal{L}$ and $\mathcal{F}$ should be read downwards in parallel, $\mathcal{L}$ before $\mathcal{F}$. Figure 6 shows a bottom-up process generating the string *What beans did Mary say that John ate?* from the dependency description of Figure 5. The columns $\mathcal{M}$ and $\mathcal{S}$ should be read upwards in parallel, $\mathcal{M}$ before $\mathcal{S}$.

3.4 String Merging

We have increased the expressive power of our dependency grammars by modifying the $\mathcal{S}$ variables, i.e., by extending the adjoin operation. In the first version, the adjoin operation randomly permutes the node label and the strings of the child nodes, and concatenates the result. In the second version, it randomly inserts the strings of the child nodes, and any moved strings to be realized at the current node, into the node label.

The adjoin operation can be further refined to allow handling an even wider range of phenomena, such as negation in French. Here, the dependent string is merged with the label of the regent, as *ne ... pas* is wrapped around portions of the verb phrase, e.g., *Ne me quitte pas!*, see (Brel, 1959). Figure 7 shows a dependency tree for this. In addition to this, the node labels may be linguistic abstractions, e.g. "negation", calling on the $\mathcal{S}$ variables also for their surface-string realization.

Note that the expressive power of the grammar depends on the possible distributions of the string probabilities $P_{\mathcal{S}}$. Since each node label can be moved and realized at the root node, any language can be recognized to which the string probabilities allow assigning the entire probablity mass, and the grammar will possess at least this expressive power.

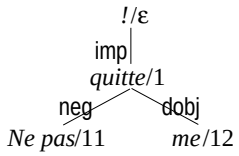

Figure 7: Dependency tree for *Ne me quitte pas!*

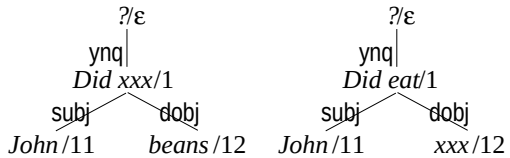

Figure 8: Dependency trees for *Did John xxx beans?* and *Did John eat xxx?*

4 A Computational Rendering

A close approximation of the described stochastic model of dependency syntax has been realized as a type of probabilistic bottom-up chart parser.

4.1 Model Specialization

The following modifications, which are really just specializations, were made to the proposed model for efficiency reasons and to cope with sparse data.

According to Tesnière, a nucleus is a unit that contains both the syntactic and semantic head and that does not exhibit any internal syntactic structure. We take the view that a nucleus consists of a content word, i.e., an open-class word, and all function words adding information to it that could just as well have been realized morphologically. For example, the definite article associates definiteness with a word, which could just has well have been manifested in the word form, as it is done in North-Germanic languages; a preposition could be realized as a locational or temporal inflection, as is done in Finnish. The longest nuclei we currently allow are verb chains of the form *that have been eaten*, as in *John knows that the beans have been eaten*.

The $\mathcal{F}$ variables were decomposed into generating the set of obligatory arguments, i.e., the valency or lexical complement, at once, as in the original model. Optional modifiers (adjuncts) are attached through one memory-less process for each modifier type, resulting in geometric distributions for these. This is the same separation of arguments and adjuncts as that employed by (Collins, 1997). However, the $\mathcal{L}$ variables remained as described above, thus leaving the lexical collocation statistics intact.

The movement probability was divided into three parts: the probability of moving the string of a particular argument dependent from its regent, that of a moved dependency type passing through a particular other dependency type, and that of a dependency type landing beneath a particular other dependency type. The one type of movement that is not yet properly handled is assigning arguments and adjuncts to dislocated heads, as in *What book did John read by Chomsky?*

The string-realization probability is a straightforward generalization of that given at the end of Section 3.1, and they are defined through regular expressions. Basically, each unmoved dependent string, each moved string landed at the current node, and each token of the nucleus labeling the current node are treated as units that are randomly permuted. Whenever possible, strings are generalized to their dependency types, but accurately modelling dependent order in French requires inspecting the actual strings of dependent clitics. Open-class words are typically generalized to their word class. String merging only applies to a small class of nuclei, where we treat the individual tokens of the dependent string, which is typically its label, as separate units when performing the permutation.

4.2 The Chart Parser

The parsing algorithm, which draws on the Cocke-Kasami-Younger (CKY) algorithm, see (Younger, 1967), is formulated as a probabilistic deduction scheme, which in turn is realized as an agenda-driven chart-parser. The top-level control is similar to that of (Pereira and Shieber, 1987), pp. 196–210. The parser is implemented in Prolog, and it relies heavily on using set and bag operations as primitives, utilizing and extending existing SICStus libraries.

The parser first nondeterministically segments the input string into nuclei, using a lexicon, and each possible nucleus spawns edges for the initial chart. Due to discontinuous nuclei, each edge spans not a single pair of string positions, indicating its start and end position, but a set of such string-position pairs, and we call this set an index. If the index is a singleton set, then it is continuous. We extend the notion of adjacent indices to be any two non-overlapping indices where one has a start position that equals an end position of the other.

The lexicon contains information about the roles (dependency types linking it to its regent) and valencies (sets[3] of types of argument dependents) that are possible for each nucleus. These are hard constraints. Unknown words are included in nuclei in a judicious way and the resulting nuclei are assigned all reasonable role/valency pairs in the lexicon. For example, the parser "correctly" analyzes the sentences *Did John xxx beans?* and *Did John eat xxx?* as shown in Figure 8, where *xxx* is not in the lexicon.

For each edge added to the initial chart, the lexicon predicts a single valency, but a set of alternative roles. Edges are added to cover all possible valen-

[3] Due to the uniqueness principle of arguments, these are sets, rather than bags.

cies for each nucleus. The roles correspond to the "goal" of dotted items used in traditional chart parsing, and the unfilled valency slots play the part of the "body", i.e., the portion of the RHS following the dot that remains to be found. If an argument is attached to the edge, the corresponding valency slot is filled in the resulting new edge; no argument can be attached to an edge unless there is a corresponding unfilled valency slot for it, or it is licensed by a moved argument. For obvious reasons, the lexicon cannot predict all possible combinations of adjuncts for each nucleus, and in fact predicts none at all. There will in general be multiple derivations of any edge with more than one dependent, but the parser avoids adding duplicate edges to the chart in the same way as a traditional chart parser does.

The parser employs a packed parse forest (PPF) to represent the set of all possible analyses and the probability of each analysis is recoverable from the PPF entries. Since optional modifiers are not predicted by the lexicon, the chart does not contain any edges that correspond directly to passive edges in traditional chart parsing; at any point, an adjunct can always be added to an existing edge to form a new edge. In some sense, though, the PPF nodes play the role of passive edges, since the parser never attempts to combine two edges, only one edge and one PPF node, and the latter will always be a dependent of the former, directly, or indirectly through the lists of moved dependents. The edge and PPF node to be combined are required to have adjacent indices, and their union is the index of the new edge.

The main point in using a packed parse forest is to perform local ambiguity packing, which means abstracting over differences in internal structure that do not matter for further parsing. When attching a PPF node to some edge as a direct or indirect dependent, the only relevant features are its index, its nucleus, its role and its moved dependents. Other features necessary for recovering the complete analysis are recorded in the PPF entries of the node, but are not used for parsing.

To indicate the alternative that no more dependents are added to an edge, it is converted into a set of PPF updates, where each alternative role of the edge adds or updates one PPF entry. When doing this, any unfilled valency slots are added to the edge's set of moved arguments, which in turn is inherited by the resulting PPF update. The edges are actually not assigned probabilities, since they contain enough information to derive the appropriate probabilities once they are converted into PPF entries. To avoid the combinatorial explosion of unrestricted string merging, we only allow edges with continuous indices to be converted into PPF entries, with the exception of a very limited class of lexically signaled nuclei, such as the *ne pas*, *ne jamais*, etc., scheme of French negation.

4.3 Pruning

As opposed to traditional chart parsing, meaningful upper and lower bounds of the supply and demand for the dependency types of the "goal" (roles) and "body" (valency) of each edge can be determined from the initial chart, which allows performing sophisticated pruning. The basic idea is that if some edge is proposed with a role that is not sought outside its index, this role can safely be removed. For example, the word *me* could potentially be an indirect object, but if there is no other word in the input string that can have an indirect object as an argument, this alternative can be discarded.

This idea is generalized to a variant of pigeonhole reasoning, in the vein of

> If we select this role or edge, then there are by necessity too few or too many of some dependency type sought or offered in the chart.

or alternatively

> If we select this nucleus or edge, then we cannot span the entire input string.

Pruning is currently only applied to the initial chart to remove logically impossible alternatives and used to filter out impossible edges produced in the prediction step. Nonetheless, it reduces parsing times by an order of magnitude or more for many of the test examples. It would however be possible to apply similar ideas to intermittently remove alternatives that are known to be suboptimal, or to heuristically prune unlikely search branches.

5 Discussion

We have proposed a generative, statistical theory of dependency syntax, based on Tesnière's classical theory, that models crossing dependency links, discontinuous nuclei and string merging. The key insight was to separate the tree-generation and string-realization processes. The model has been realized as a type of probabilistic chart parser. The only other high-fidelity computational rendering of Tesnière's dependency syntax that we are aware of is that of (Tapanainen and Järvinen, 1997), which is neither generative nor statistical.

The stochastic model generating dependency trees is very similar to other statistical dependency models, e.g., to that of (Alshawi, 1996). Formulating it using Gorn's notation and the $\mathcal{L}$ and $\mathcal{F}$ variables, though, is concise, elegant and novel. Nothing prevents conditioning the random variables on arbitrary portions of the partial tree generated this far, using, e.g., maximum-entropy or decision-tree models to extract relevant features of it; there is no difference

in principle between our model and history-based parsing, see (Black et al., 1993; Magerman, 1995).

The proposed treatment of string realization through the use of the $\mathcal{S}$ and $\mathcal{M}$ variables is also both truly novel and important. While phrase-structure grammars overemphasize word order by making the processes generating the $\mathcal{S}$ variables deterministic, Tesnière treats string realization as a secondary issue. We find a middle ground by using stochastic processes to generate the $\mathcal{S}$ and $\mathcal{M}$ variables, thus reinstating word order as a parameter of equal importance as, say, lexical collocation statistics. It is however not elevated to the hard-constraint status it enjoys in phrase-structure grammars.

Due to the subordinate role of string realization in classical dependency grammar, the technical problems related to incorporating movement into the string-realization process have not been investigated in great detail. Our use of the $\mathcal{M}$ variables is motivated partly by practical considerations, and partly by linguistic ones. The former in the sense that this allows designing efficient parsing algorithms for handling also crossing dependency links. The latter as this gives us a quantitative handle on the empirically observed resistance against crossing dependency links. As Tesnière points out, there is locality in string realization in the sense that dependents tend to be realized adjacent to their regents. This fact is reflected by the model parameters, which also model, probabilistically, barrier constraints, constraints on landing sites, etc. It is noteworthy that treating movement as in GPSG, with the use of the "slash" feature, see (Gazdar et al., 1985), pp. 137–168, or as is done in (Collins, 1997), is the converse of that proposed here for dependency grammars: the former pass constituents down the tree, the $\mathcal{M}$ variables pass strings up the tree.

The relationship between the proposed stochastic model of dependency syntax and a number of other prominent stochastic grammars is explored in detail in (Samuelsson, 2000).

References

Hiyan Alshawi. 1996. Head automata and bilingual tiling: Translation with minimal representations. *Procs. 34th Annual Meeting of the Association for Computational Linguistics*, pages 167–176.

Ezra Black, Fred Jelinek, John Lafferty, David Magerman, Robert Mercer, and Salim Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pages 31–37.

Jacques Brel. 1959. Ne me quitte pas. *La Valse à Mille Temps (PHI 6325.205)*.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. *Procs. 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

Ulrich Engel, 1996. *Tesnière Mißverstanden: Lucien Tesnière – Syntaxe Structurale et Operation Mentales. Akten des deutsch-französischen Kolloquiums anläßlich der 100 Wiederkehr seines Geburstages, Strasbourg 1993*, volume 348 of Linguistische Arbeiten, pages 53–61. Niedermeyer, Tübingen.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Basil Blackwell Publishing, Oxford, England. Also published by Harvard University Press, Cambridge, MA.

Saul Gorn. 1962. Processors for infinite codes of shannon-fano type. *Symp. Math. Theory of Automata*.

David Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

Timo Järvinen. 1998. *Tesniére's Structural Syntax Reworked*. University of Helsinki, Helsinki.

David Magerman. 1995. Statistical decision-tree models for parsing. *Procs. 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283.

Igor Mel'čuk. 1987. *Dependency Syntax*. State University of New York Press, Albany.

Fernando Pereira and Stuart Shieber. 1987. *Prolog and Natural-Language Analysis*. CSLI Lecture Note 10.

Jane Robinson. 1970. Dependency structures and transformational rules. *Language*, 46:259–285.

Christer Samuelsson. 2000. A theory of stochastic grammars. In *Proceedings of NLP-2000*, pages 92–105. Springer Verlag.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguations by means of tree-grammars. *Procs. 16th International Conference on Computational Linguistics*, at the very end.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. *Procs. 5th Conference on Applied Natural Language Processing*, pages 64–71.

Lucien Tesnière. 1959. *Éléments de Syntaxe Structurale*. Libraire C. Klincksieck, Paris.

David H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.