Learning and parsing stochastic unification-based grammars

Mark Johnson

Cognitive and Linguistic Sciences and Computer Science
Brown University
Providence RI 02912, USA
Mark_Johnson@Brown.edu

Abstract. Stochastic Unification-Based Grammars combine knowledge-rich and data-rich approaches to natural language processing. This provides a rich structure to the learning and parsing (decoding) tasks that can be described with undirected graphical models. While most work to date has treated parsing as a straight-forward multi-class classification problem, we are beginning to see how this structure can be exploited in learning and parsing. Exploiting this structure is likely to become more important as the research focus moves from parsing to more realistic tasks such as machine translation and summarization.

1 Introduction

This paper summarizes recent research into Stochastic Unification-Based Grammars (SUBGs), which attempt to combine linguistic insights expressed by grammars with modern machine-learning techniques. So far most work has focused on parsing, i.e., identifying the syntactic and semantic structure of sentences, and this paper focuses on this application. But the community is beginning to investigate applications of these techniques such as automatic summarization [1] and machine translation, which are more challenging and present interesting opportunities for machine learning.

A *parse* consists of a string of words, called its *yield*, together with its syntactic structure and/or semantics. A syntactic structure describes how words are organized into phrases and clauses, while the semantics is the meaning of a sentence. While trees are most commonly used to represent syntactic structure, the linguistic theories discussed in this paper use both trees and attribute-value structures to represent the syntactic and semantic properties of a sentence. Even though modern linguistics has been investigating syntax and semantics since the mid-1950s and before, there are still large gaps in our knowledge of English and other languages, and considerable dispute over exactly what the structure of particular sentences actually is.

The parsing problem is to identify the parse of a sentence intended by its speaker or author. Undoubtedly this depends on a wide variety of factors, including the discourse context and world knowledge, but most work on statistical

parsing ignore these. Not surprisingly, ambiguity is a pervasive problem in parsing: many different parses have the same yield (e.g., consider sentences such as "I saw the man with the T.V."). Currently statistical and machine-learning approaches provide the most systematic way to identify the most likely intended parse.

The basic assumption behind the work described here is that linguists have identified reasonably well the general principles that determine the possible structures of a language, but they have not identified a satisfactory method for disambiguating sentences, so this must be inferred from training data. By contrast, most work on statistical parsing seems to assume that linguists can only accurately determine the structures of particular sentences (specifically, those in the treebank training data), and only incorporates much weaker linguistic assumptions [2, 3].

SUBGs define probability distributions over the parses defined by Unification-Based Grammars (UBGs). These grammars can incorporate virtually all kinds of linguistically important constraints. There are several different kinds of UBGs, including Head-Driven Phrase-Structure Grammar (HPSG) [4, 5] and Lexical-Functional Grammar (LFG) [6, 7]. This paper uses LFG, but the approach is general and can be used with virtually any UBG. Section 2 describes LFG representations in some more detail, and section 4 explains how probability distributions are defined over these structures.

We chose LFG because two decades of work have produced in broad-coverage grammars [8] and well-developed non-statistical parsing technology [9, 10]. In particular, there are parsing algorithms that produce "packed" representations of a set of parses which can often find an exponential number of different parses for a sentence in polynomial time and represent them in polynomial space. Section 5 outlines dynamic programming algorithms for estimating statistical models and identifying the most probable parse from the set of parses encoded by a packed representation.

The notation used in this paper is as follows. Variables are written in upper case italic, e.g., X, Y, etc., the sets they range over are written in script, e.g., $\mathcal{X}, \mathcal{Y}$, etc., while specific values are written in lower case italic, e.g., x, y, etc. In the case of vector-valued entities, subscripts indicate particular components.

2 Lexical-Functional Grammar

An LFG parse describes the different aspects of a sentence's syntactic structure and semantics using several different structures; for simplicity we only discuss two of these here. The constituent or c-structure of a parse is a tree that describes how words and phrases combine to form larger syntactic units. The functional or f-structure is a particular kind of directed graph known as an attribute-value structure that identifies the function-argument dependencies of the parse.

Figure 1 depicts the c-structure and f-structure for the sentence "Alex promised Sasha to leave". The fact that the single phrase "Alex" fills two functional roles in this sentence (it is the subject of "promise" as well as the subject of "leave") is

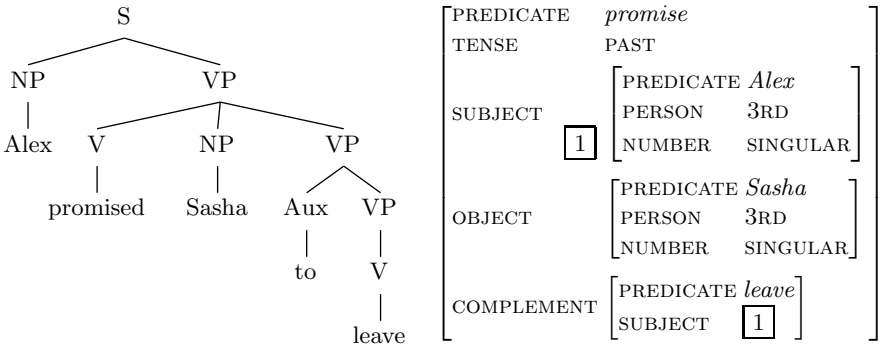

Fig. 1. The c-structure and f-structure for "Alex promised Sasha to leave".

not represented in the c-structure tree, but it is represented by the re-entrancy in the f-structure (which is depicted by co-indexation in the figure). LFG parses use similar f-structure re-entrancies to encode long-distance dependencies in phrases such as "Sam bought the program that Alex wants to use", where "the program" is both the object of "bought" and of "use".

The precise way in which grammars specify how strings are associated with parses differs in different kinds of UBGs, and is not important for what follows. Let Ω be the set of parses for the sentences of a language, and for each parse $\omega \in \Omega$ let $Y(\omega)$ be its yield. If y is a string then $\Omega(y) = \{\omega \in \Omega | Y(\omega) = y\}$ is the set of parses with yield y. We require that $\Omega(y)$ be a finite for all y; this is almost always true of (bug-free) UBGs. For our LFGs $|\Omega(y)|$ ranges from one to the hundreds of thousands.

It important for the dynamic programming algorithms described below that each parse ω consist of a finite set of parse *fragments*.[1] Precisely what a fragment is is unimportant, but typically fragments are units directly or indirectly defined by the grammar (e.g., chart edges, local trees, attribute-value pairs, etc.) out of which parses are constructed. For example, in Figure 1 one fragment might consists of all of the information associated with the word "Alex", i.e., the NP c-structure node and the f-structure node with the index "1". This fragment would also appear in other parses containing the word "Alex". This sharing of fragments is key to the dynamic programming algorithms described below. If y is a string let $\mathcal{F}(y) = \bigcup_{\omega \in \Omega(y)} \{f \in \omega\}$ be the set of fragments appearing in any parse of y. $\mathcal{F}(y)$ is finite since $\Omega(y)$ and each $\omega \in \Omega(y)$ are finite.

3 Maxwell-Kaplan packed parse representations

As mentioned earlier, ambiguity is a pervasive problem in parsing, and much of the work on parsing over the past several decades has focused on methods for effi-

[1] Fragments were called "features" in [11], but we use the word "fragment" here to avoid confusion with the machine learning use of the word "feature".

ciently finding and compactly representing the parses of a sentence. Maxwell and Kaplan [9, 10] developed a dynamic programming framework for UBG parsing that seems to be more general than other approaches. The dynamic programming algorithms described below take the packed representations produced by their parsing algorithm as input, so we describe them here. The intuition motivating the Maxwell-Kaplan packed representations is that for most strings y, many of the fragments in $\mathcal{F}(y)$ occur in many of the parses $\Omega(y)$. This is often the case in natural language, since the same substructure can appear as a component of many different parses.

Maxwell-Kaplan packed representations are defined in terms of conditions on the values assigned to a vector of auxiliary variables X. These variables have no direct linguistic interpretation; rather, each different assignment of values to these variables identifies a set of fragments which constitutes one of the parses represented by the packed representation. A *condition* on X is a function from $\mathcal{X}$ (the range of X) to $\{0, 1\}$. While for uniformity we write conditions as functions on the entire vector X, in practice the Maxwell-Kaplan parsing algorithm usually produces conditions whose value depends only on a few of the variables in X, and the efficiency of the algorithms described here depends on this.

A *packed representation* of a finite set of parses is a quadruple $R = (\mathcal{F}', X, N, \alpha)$, where:

- $\mathcal{F}' \supseteq \mathcal{F}(y)$ is a finite set of fragments,
- X is a finite vector of *variables*, where each variable X_ℓ ranges over the finite set $\mathcal{X}_\ell$,
- N is a finite set of conditions on X called the *no-goods*,[2] and
- α is a function that maps each fragment $f \in \mathcal{F}'$ to a condition α_f on X.

A vector of values x *satisfies the no-goods* N iff $N(x) = 1$, where $N(x) = \prod_{\eta \in N} \eta(x)$. Each x that satisfies the no-goods *identifies* a parse $\omega(x) = \{f \in \mathcal{F}' \big| \alpha_f(x) = 1\}$, i.e., ω is the set of features whose conditions are satisfied by x. We require that each parse be identified by a *unique* value satisfying the no-goods. That is, we require that:

$$\forall x, x' \in \mathcal{X} \text{ if } N(x) = N(x') = 1 \text{ and } \omega(x) = \omega(x') \text{ then } x = x' \qquad (1)$$

A packed representation R *represents* the set of parses $\Omega(R)$ that are identified by values that satisfy the no-goods, i.e., $\Omega(R) = \{\omega(x) | x \in \mathcal{X}, N(x) = 1\}$.

Maxwell and Kaplan describe a parsing algorithm for unification-based grammars that takes as input a string y and returns a packed representation R such that $\Omega(R) = \Omega(y)$, i.e., R represents the set of parses of the string y. The SUBG parsing and estimation algorithms described here use the Maxwell-Kaplan parsing algorithm as a subroutine.

[2] The name "no-good" comes from the TMS literature, and was used by Maxwell and Kaplan. However, here the no-goods actually identify the *good* variable assignments.

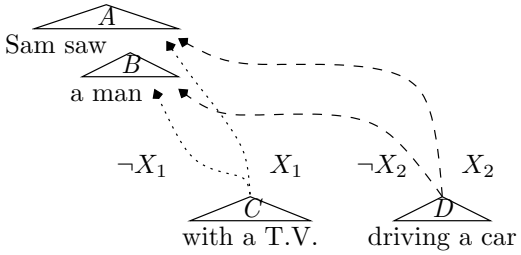

Fig. 2. A depiction of a packed parse forest encoding possible parses of "Sam saw a man with a T.V. driving a car".

Example. Figure 2 depicts in extremely schematic form a packed parse representation for the sentence "Sam saw a man with a T.V. driving a car". This admittedly rather contrived sentence contains two interacting ambiguities.[3] First, "with a T.V." can either modify "saw" or "a man" (i.e., either Sam used the T.V., or the man had the T.V.). The boolean variable X_1 encodes which phrase "with a T.V." modifies.

Second, "driving a car" can also either modify "saw" or "a man" (either the seeing was done while driving, or else the man was driving a car; the semantically implausible reading where the T.V. was driving a car is not represented). The boolean variable X_2 encodes which phrase "driving a car" modifies.

Many linguists would claim that not all combinations of these ambiguities are possible. Specifically, if "with a T.V." modifies "saw" then "driving a car" cannot modify "a man", since that would result in a crossing dependency of a kind not found in English. This constraint is encoded by the no-good $\neg X_1 \vee X_2$.

Suppose that ξ_A is the parse fragment for the phrase "Sam saw", ξ_B is the fragment for "a man", ξ_C for "with a T.V." and ξ_D for "driving a car". Let ξ_{CA} be the parse fragment representing the attachment of ξ_C to ξ_A (i.e., the c-structure and f-structure elements representing the attachment of "with a T.V." to "Sam saw"), and let ξ_{CB}, ξ_{DA}, ξ_{DB} be the other corresponding attachment fragments. Then the following schematic packed representation encodes the possible parses:

$$\begin{aligned}
\mathcal{F}' &= \{\xi_A, \xi_B, \xi_C, \xi_D, \xi_{CA}, \xi_{CB}, \xi_{DA}, \xi_{DB}\} \\
X &= (X_1, X_2) \\
N &= \{\neg X_1 \vee X_2\} \\
&\quad \alpha(\xi_{CA}) = X_1, \alpha(\xi_{CB}) = \neg X_1, \alpha(\xi_{DA}) = X_2, \alpha(\xi_{DB}) = \neg X_2, \\
&\quad \alpha(\xi_A) = \alpha(\xi_B) = \alpha(\xi_C) = \alpha(\xi_D) = 1.
\end{aligned}$$

In LFGs and similiar grammars the fragments containing lexical items (i.e., words) are usually very large, since these grammars typically contain a lot of

[3] Interestingly, most of the ambiguities that cause so much trouble in parsing are extremely difficult for humans to detect.

lexical information. Notice that the fragments $\xi_A, \ldots, \xi_D$ corresponding to the phrases in the sentence occur only once in the packed representation, rather than appearing three times, as they would if the parses were enumerated. In the packed representation these fragments are associated with the constant condition 1, which indicates that they all appear in every parse. Only the fragments required to connect up these phrasal fragments have non-constant conditions, indicating that they are the only fragments that vary across the parses of this sentence. Such structure sharing, especially of lexical information, is a goal of the Maxwell-Kaplan approach.

4 Stochastic Unification-Based Grammars

SUBGs use exponential models (also known as log-linear or MaxEnt models or Markov Random Fields) extend UBGs by defining probability distributions over the parses of a UBG. Abney [12] proposed the use of exponential models for defining probability distributions over the parses of a UBG. Johnson et al. [13] noted that calculating the partition function and the various expectations required for estimation (see below) is computationally very expensive because they require integrating over all parses of all sentences. However, Johnson et al. pointed out that parsing and related tasks only require the conditional distribution of parses given their yields, and that these conditional distributions are much less computationally demanding to estimate.

A SUBG is a triple (U, g, θ), where U is a unification grammar that defines a set Ω of parses as described above, $g = (g_1, \ldots, g_m)$ is a vector of features, and $\theta = (\theta_1, \ldots, \theta_m)$ is a vector of non-negative real-valued parameters called *feature weights*. A *feature* is a real-valued function of parses Ω.[4]

If we are prepared to enumerate all of the parses of the sentences in our training and test data then features can be any real-valued function of parses whatsoever. However, the dynamic programming algorithms described below require the information encoded in features to be local with respect to the fragments $\mathcal{F}$ of packed parse representations. Specifically, they require that features be linear in terms of the fragments of parses, i.e., each fragment $f \in \mathcal{F}$ is associated with a finite vector of real values $(g_1(f), \ldots, g_m(f))$ which define the feature functions for parses as follows:

$$g_k(\omega) = \sum_{f \in \omega} g_k(f), \text{ for } k = 1 \ldots m. \tag{2}$$

This requires features be very local with respect to fragments, which gives up the ability to define features arbitrarily. Note that we can encode essentially arbitrary non-local information in the attributes of the underlying unification grammar and then define features locally in terms of those attributes.

[4] Features were called "properties" in [11, 13] in order to avoid confusion with the linguistic use of the word "feature".

We now describe how SUBGs define conditional probability distributions over parses given yields. The conditional probability $\mathrm{P}_\theta(\omega|y)$ of a parse $\omega \in \Omega(y)$ given a yield y is:

$$\mathrm{P}_\theta(\omega|y) = \frac{W_\theta(\omega)}{Z_\theta(y)}, \text{ where } W_\theta(\omega) = \prod_{j=1}^{m} \theta_j^{g_j(\omega)}, \text{ and } Z_\theta(y) = \sum_{\omega' \in \Omega(y)} W_\theta(\omega').$$

Intuitively, if $g_j(\omega)$ is the number of times that feature j occurs in ω then θ_j is the weight or cost of each occurrence of feature j, and $Z_\theta(y)$ is a normalising constant that ensures that the probability of all parses of y sum to 1.

Such a distribution can be used for a variety of tasks. The most straightforward application is parsing: the most likely parse of a string of words y is:

$$\hat{\omega}(y) = \operatorname*{argmax}_{\omega \in \Omega(y)} \mathrm{P}(\omega|y) = \operatorname*{argmax}_{\omega \in \Omega(y)} W_\theta(\omega).$$

$\hat{\omega}(y)$ can be found by exhaustively enumerating $\Omega(y)$. But even though $\Omega(y)$ is finite, with large grammars and long sentences finding the most likely parse by direct enumeration can be extremely time-consuming. The dynamic programming algorithm described below exploits the structure of packed representations to avoid this exhaustive search.

Now we turn to estimation. Suppose we have a training corpus D of sentences y_i and their correct parses ω_i, i.e., $D = ((y_1, \omega_1), \ldots, (y_n, \omega_n))$, from which we wish to estimate the feature weights θ.[5] The $\hat{\theta}$ that maximizes the *conditional likelihood* $L_D(\theta)$ is a consistent estimator of the conditional distribution.[6]

$$\hat{\theta} = \operatorname*{argmax}_{\theta} L_D(\theta) = \prod_{i=1}^{n} \mathrm{P}_\theta(\omega_i|y_i)$$

Practial methods for optimizing L_D, such as the Conjugate Gradient and L-BFGS algorithms as well as the various iterative scaling algorithms, require the partial derivatives of L_D as well as L_D itself.

$$\frac{\partial L_D}{\partial \theta_k} = \frac{L_D(\theta)}{\theta_k} \sum_{i=1}^{n} \left(g_k(\omega_i) - E_\theta[g_k|y_i] \right), \text{where:}$$

$$E_\theta[g|y] = \sum_{\omega \in \Omega(y)} g(\omega) \mathrm{P}_\theta(\omega|y) = \sum_{\omega \in \Omega(y)} \frac{g(\omega) W_\theta(\omega)}{Z_\theta(y)}$$

Here $E_\theta[g|y]$ is the *conditional expectation* of g with respect to distribution P_θ over parses $\Omega(y)$ of the sentence y. As with parsing, computing this conditional expectation by explicit summation over $\Omega(y)$ can be computationally expensive.

[5] We call D *fully labelled* because it identifies the correct parse ω_i for each sentence $y_i = Y(\omega_i)$ in D; see [14] for conditional estimation from partially labelled data.

[6] In fact [13] and later work estimate θ by maximizing a regularized conditional likelihood obtained by multiplying L_D by a zero-mean diagonal-variance Gaussian.

5 Packed representations and graphical models

The previous section treated parses in SUBGs as atomic (i.e., the features are functions of the entire parse), whereas they actually have a rich internal structure. As explained earlier the Maxwell-Kaplan parsing algorithm exploits the sharing of fragments in parses to produce a "packed representation" of a set of parses. This section describes explains how dynamic programming algorithms for graphical models can be used to compute the most likely parse and the expected value of features directly from these packed representations [11].

These methods are analogues of the well-known dynamic programming algorithms for Probabilistic Context-Free Grammars (PCFGs); specifically the Viterbi algorithm for finding the most probable parse of a string and the Inside-Outside algorithm for estimating a PCFG from unparsed training data (which we use to calculate the expected number of times each feature occurs). In fact, because the Maxwell-Kaplan packed representations are just Truth Maintenance System (TMS) representations [15], the statistical techniques described here should extend to non-linguistic applications of TMSs as well.

Dynamic programming techniques have been applied to log-linear models in other setting. Lafferty et al. show how dynamic programming can be used to compute the statistics required for conditional estimation of log-linear models of labeled sequences where the properties can include arbitrary functions of the input string [16]. The closest work we know of to the approach described here is that of Miyao, Tsujii and colleagues [17, 18]. They also describe a technique for calculating the statistics required to estimate a log-linear model from packed parse forests.

The previous section introduced several important quantities for parsing and maximum likelihood estimation of SUBGs. In each case we show that the quantity can be expressed as the value that maximises a product of functions or else as the sum of a product of functions, each of which depends on a small subset of the variables X of a packed parse representation. These are the kinds of quantities for which dynamic programming algorithms for graphical models have been developed.

As explained in section 4, the most probable parse is $\hat{\omega}(y) = \operatorname{argmax}_{\omega \in \Omega(y)} W_\theta(\omega)$. Given a packed representation $(\mathcal{F}', X, N, \alpha)$ for the parses $\Omega(y)$, let $\hat{x}(y)$ be the x that identifies $\hat{\omega}(y)$, i.e., $\hat{\omega}(y) = \{f \in \mathcal{F}' | \, \alpha_f(\hat{x}(y)) = 1\}$. Since $W_\theta(\hat{\omega}(y)) > 0$, it can be shown that:

$$\begin{aligned}
\hat{x}(y) &= \operatorname*{argmax}_{x \in \mathcal{X}} N(x) \prod_{j=1}^{m} \theta_j^{g_j(\omega(x))} \\
&= \operatorname*{argmax}_{x \in \mathcal{X}} N(x) \prod_{j=1}^{m} \theta_j^{\sum_{f \in \omega(x)} g_j(f)} \\
&= \operatorname*{argmax}_{x \in \mathcal{X}} N(x) \prod_{j=1}^{m} \theta_j^{\sum_{f \in \mathcal{F}'} \alpha_f(x) g_j(f)}
\end{aligned}$$

$$
\begin{aligned}
&= \operatorname*{argmax}_{x \in \mathcal{X}} N(x) \prod_{j=1}^{m} \prod_{f \in \mathcal{F}'} \theta_j^{\alpha_f(x) g_j(f)} \\
&= \operatorname*{argmax}_{x \in \mathcal{X}} N(x) \prod_{f \in \mathcal{F}'} \left(\prod_{j=1}^{m} \theta_j^{g_j(f)} \right)^{\alpha_f(x)} \\
&= \operatorname*{argmax}_{x \in \mathcal{X}} \prod_{\eta \in N} \eta(x) \prod_{f \in \mathcal{F}'} h_{\theta,f}(x) \qquad (3)
\end{aligned}
$$

where $h_{\theta,f}(x) = \prod_{j=1}^{m} \theta_j^{g_j(f)}$ if $\alpha_f(x) = 1$ and $h_{\theta,f}(x) = 1$ if $\alpha_f(x) = 0$. That is, $h_{\theta,f}(x)$ is the weight of the fragment f in the parse $\omega(x)$. Note that $h_{\theta,f}(x)$ depends on exactly the same subset of variables in X as α_f does. As (3) makes clear, finding $\hat{x}(y)$ involves maximising a product of functions where each function depends on a subset of the variables X. As explained below, this is exactly the kind of maximisation that can be solved using graphical model techniques.

We now turn to the partition function. As mentioned earlier, calculating the partition function $Z_\theta(y) = \sum_{\omega \in \Omega(y)} W_\theta(\omega)$ by explicitly summing over all parses $\Omega(y)$ can be computationally expensive. However, there is an alternative method for calculating $Z_\theta(y_i)$ that does not involve this enumeration. As noted above, for each yield $y_i, i = 1, \ldots, n$, the Maxwell-Kaplan algorithm returns a packed parse R_i that represents the parses of y_i, i.e., $\Omega(y_i) = \Omega(R_i)$. A derivation parallel to the one for (3) shows that for $R = (\mathcal{F}', X, N, \alpha)$:

$$
Z_\theta(\Omega(R)) = \sum_{x \in \mathcal{X}} \prod_{\eta \in N} \eta(x) \prod_{f \in \mathcal{F}'} h_{\theta,f}(x) \qquad (4)
$$

This derivation relies on the isomorphism between parses and variable assignments in (1). It turns out that sums of products of this kind can also be calculated using graphical model techniques.

Similar remarks apply to the computation of the conditional expectations $E_\theta[g_k|y_i]$. Again, let $R = (\mathcal{F}', X, N, \alpha)$ be a packed representation such that $\Omega(R) = \Omega(y_i)$. First, note that (2) implies that:

$$
E_\theta[g_k|y_i] = \sum_{f \in \mathcal{F}'} g_k(f)\ \mathrm{P}(\{\omega : f \in \omega\}|y_i).
$$

Note that $\mathrm{P}(\{\omega : f \in \omega\}|y_i)$ involves the sum of weights over all $x \in \mathcal{X}$ subject to the conditions that $N(x) = 1$ and $\alpha_f(x) = 1$. Thus $\mathrm{P}(\{\omega : f \in \omega\}|y_i)$ can also be expressed in a form that is easy to evaluate using graphical techniques.

$$
Z_\theta(\Omega(R))\mathrm{P}_\theta(\{\omega : f \in \omega\}|y_i) = \sum_{x \in \mathcal{X}} \alpha_f(x) \prod_{\eta \in N} \eta(x) \prod_{f' \in \mathcal{F}'} h_{\theta,f'}(x) \qquad (5)
$$

5.1 Graphical model calculations

In this subsection we briefly review graphical model algorithms for maximizing and summing products of functions of the kind presented above. The quantities

(3), (4) and (5) involve maximisation or summation over a product of functions, each of which depends only on the values of a subset of the variables X. There are dynamic programming algorithms for calculating all of these quantities, but for reasons of space we only describe an algorithm that finds the most probable parse $\hat{\omega}(y)$ of a string y. As explained above, this is equivalent to finding the $x \in \mathcal{X}$ that maximizes (3).

It turns out that the algorithm for maximisation is a generalisation of the Viterbi algorithm for HMMs, and the algorithm for computing the summation in (5) is a generalisation of the forward-backward algorithm for HMMs [19]. Viewed abstractly, these algorithms simplify these expressions by moving common factors over the max or sum operators respectively. These techniques are now relatively standard; the most well-known approach involves junction trees [20, 21]. We adopt the approach approach described by [22], which is a straightforward generalization of HMM dynamic programming with minimal assumptions and programming overhead.

To explain the algorithm we use the following notation. If x and x' are both vectors of length m then $x =_j x'$ iff x and x' disagree on at most their jth components, i.e., $x_k = x'_k$ for $k = 1, \ldots, j-1, j+1, \ldots m$. If f is a function whose domain is $\mathcal{X}$, we say that f *depends on* the set of variables $d(f) = \{X_j | \exists x, x' \in \mathcal{X}, x =_j x', f(x) \neq f(x')\}$. That is, $X_j \in d(f)$ iff changing the value of X_j can change the value of f.

The algorithm relies on the fact that the variables in $X = (X_1, \ldots, X_n)$ are ordered (e.g., X_1 precedes X_2, etc.), and while the algorithm is correct for any variable ordering, its efficiency may vary dramatically depending on which ordering is chosen. Let $\mathcal{H}$ be any set of functions whose domains are X. We partition $\mathcal{H}$ into disjoint subsets $\mathcal{H}_1, \ldots, \mathcal{H}_{n+1}$, where $\mathcal{H}_j$ is the subset of $\mathcal{H}$ that depend on X_j but do not depend on any variables ordered before X_j, and $\mathcal{H}_{n+1}$ is the subset of $\mathcal{H}$ that do not depend on any variables at all (i.e., they are constants).[7] That is, $\mathcal{H}_j = \{H \in \mathcal{H} | X_j \in d(H), \forall i < j \, X_i \notin d(H)\}$ and $\mathcal{H}_{n+1} = \{H \in \mathcal{H} | d(H) = \emptyset\}$.

In order to find the most probable parse we must find the x that maximizes the product of functions (3). Here we describe a general algorithm for finding $M_{\max}$ and $\hat{x}$ that satisfy (6-7).

$$M_{\max} = \max_{x \in \mathcal{X}} \prod_{A \in \mathcal{A}} A(x) \tag{6}$$

$$\hat{x} = \operatorname*{argmax}_{x \in \mathcal{X}} \prod_{A \in \mathcal{A}} A(x). \tag{7}$$

The procedure depends on two sequences of functions $M_i, i = 1, \ldots, n+1$ and $V_i, i = 1, \ldots, n$. Informally, M_i is the maximum value attained by the subset of the functions $\mathcal{A}$ that depend on one of the variables $X_1, \ldots, X_i$, and V_i identifies the x at which this maximum is attained.

[7] Strictly speaking this does not necessarily define a partition, as some of the subsets $\mathcal{H}_j$ may be empty.

To simplify notation we write these functions as functions of the entire set of variables X, but the efficiency of the algorithm requires that they depend on a much smaller set of variables. The M_i are real valued, while each V_i ranges over $\mathcal{X}_i$. Let $\mathcal{M} = \{M_1, \ldots, M_n\}$. $\mathcal{A}$ and $\mathcal{M}$ are both partitioned into disjoint subsets $A_1, \ldots, \mathcal{A}_{n+1}$ and $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ respectively as described above on the basis of the variables each A_i and M_i depend on.

$$M_i(x) = \max_{\substack{x' \in \mathcal{X} \\ \text{s.t. } x' =_i x}} \prod_{A \in \mathcal{A}_i} A(x') \prod_{M \in \mathcal{M}_i} M(x') \tag{8}$$

$$V_i(x) = \operatorname*{argmax}_{\substack{x' \in \mathcal{X} \\ \text{s.t. } x' =_i x}} \prod_{A \in \mathcal{A}_i} A(x') \prod_{M \in \mathcal{M}_i} M(x')$$

M_{n+1} receives a special definition, since there is no variable X_{n+1}.

$$M_{n+1} = \left(\prod_{A \in \mathcal{A}_{n+1}} A \right) \left(\prod_{M \in \mathcal{M}_{n+1}} M \right) \tag{9}$$

The $M_i(x)$ and $V_i(x)$ correspond to the intermediate quantities computed in the well-known Viterbi algorithm for HMMs. $M_i(x)$ is the maximum value of the product of terms $A \in \mathcal{A}$ that depend directly or indirectly (through some M) on the variable X_i, and the ith component of $V_i(x)$ gives the value of x_i at this maximum. That is, each M_i can be recursively expanded into the maximum of a product of functions $A \in \mathcal{A}$. If any of these functions also depend on variables ordered after X_i, M_i and V_i must also.

The definition of M_i in (8) may look circular (since M appears in the right-hand side), but in fact it is not. First, note that M_i depends only on variables ordered after X_i, so if $M_j \in \mathcal{M}_i$ then $j < i$. More specifically,

$$d(M_i) = \left(\bigcup_{A \in \mathcal{A}_i} d(A) \cup \bigcup_{M \in \mathcal{M}_i} d(M) \right) \setminus \{X_i\}.$$

Thus we can compute the M_i in the order $M_1, \ldots, M_{n+1}$, inserting M_i into the appropriate set $\mathcal{M}_k$, where $k > i$ is the next variable M_i depends on, when M_i is computed. By recursively expanding each M_i, a simple induction shows that $M_{\max} = M_{n+1}$ and that $V_i(\hat{x}) = \hat{x}_i$ (the value $\hat{x}$ assigns to X_i). Because V_i only depends on variables ordered after x_i, we can evaluate the V_i in the order $V_n, \ldots, V_1$ to find the maximising assignment $\hat{x}$.

We now briefly consider the computational complexity of this process. Clearly, the number of steps required to compute each M_i is a polynomial of order $|d(M_i)| + 1$, since we need to enumerate all possible values for the argument variables $d(M_i)$ and for each of these, maximise over the set $\mathcal{X}_i$.

Since computational effort is bounded above by a polynomial of order $|d(M_i)|+1$, we seek a variable ordering that bounds the maximum value of $|d(M_i)|$. Unfortunately, finding the ordering that minimises the maximum value of $|d(M_i)|$ is

an NP-complete problem. However, there are several efficient heuristics that are reputed in graphical models community to produce good visitation schedules. It may be that they will perform well in the SUBG parsing applications as well.

6 Conclusion

This paper described recent research in SUBGs, concentrating on dynamic programming algorithms for estimating and parsing using exponential models that do not require explicit enumeration of all parses. These algorithms are likely to play an increasingly important role as SUBGs are applied to complex natural language processing tasks such as machine translation.

The key observation is that the set of parses of a sentence possesses a non-trivial structure that can be represented using an undirected graphical model. The Maxwell-Kaplan parsing algorithm takes advantage of this structure to produce a packed representation of a set of parses, and the dynamic programming algorithms described above exploit the same structure to compute the statistics needed for parsing and estimation without enumerating all possible parses.

While almost all of the work on SUBGs has been based on exponential models, the astute reader will have noticed identifying the correct parse of a sentence is essentially just a discriminative learning task, and virtually any discriminative learning algorithm could be used. It would be interesting to see if standard discriminative learning algorithms, such as Support Vector Machines, yield better parsing performance. It would also be interesting to see if these algorithms can be trained from partially labelled data [14] and possess dynamic programming parsing and estimation algorithms.

References

1. Stefan Riezler, Tracy H. King, R.C., Zaenen, A.: Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-functional grammar. In: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, The Association for Computational Linguistics (2003) 197–204
2. Charniak, E.: A maximum-entropy-inspired parser. In: The Proceedings of the North American Chapter of the Association for Computational Linguistics. (2000) 132–139
3. Collins, M.: Three generative, lexicalised models for statistical parsing. In: The Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, San Francisco, Morgan Kaufmann (1997)
4. Pollard, C., Sag, I.A.: Information-based Syntax and Semantics. Number 13 in CSLI Lecture Notes Series. Chicago University Press, Chicago (1987)
5. Pollard, C., Sag, I.: Head-driven Phrase Structure Grammar. The University of Chicago Press, Chicago (1994)
6. Bresnan, J.: Control and complementation. In Bresnan, J., ed.: The Mental Representation of Grammatical Relations. The MIT Press, Cambridge, Massachusetts (1982) 282–390

7. Bresnan, J.: Lexical-Functional Syntax. Blackwell, Malden, MA (2001)
8. Butt, M., King, T.H., no, M.E.N., Segon, F.: A Grammar Writer's Cookbook. CSLI Publications, Stanford, CA (1999)
9. Maxwell III, J.T., Kaplan, R.M.: The interface between phrasal and functional constraints. In Dalrymple, M., Kaplan, R.M., Maxwell III, J.T., Zaenen, A., eds.: Formal Issues in Lexical-Functional Grammar. Number 47 in CSLI Lecture Notes Series. CSLI Publications (1995) 403–430
10. Maxwell III, J.T., Kaplan, R.M.: A method for disjunctive constraint satisfaction. In Dalrymple, M., Kaplan, R.M., Maxwell III, J.T., Zaenen, A., eds.: Formal Issues in Lexical-Functional Grammar. Number 47 in CSLI Lecture Notes Series. CSLI Publications (1995) 381–401
11. Geman, S., Johnson, M.: Dynamic programming for parsing and estimation of stochastic unification-based grammars. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Morgan Kaufmann (2002) 279–286
12. Abney, S.: Stochastic Attribute-Value Grammars. Computational Linguistics **23** (1997) 597–617
13. Johnson, M., Geman, S., Canon, S., Chi, Z., Riezler, S.: Estimators for stochastic "unification-based" grammars. In: The Proceedings of the 37th Annual Conference of the Association for Computational Linguistics, San Francisco, Morgan Kaufmann (1999) 535–541
14. Riezler, S., King, T.H., Kaplan, R.M., Crouch, R., Maxwell, J.T.I., Johnson, M.: Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Morgan Kaufmann (2002) 271–278
15. Forbus, K.D., de Kleer, J.: Building problem solvers. The MIT Press, Cambridge, Massachusetts (1993)
16. Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In: Machine Learning: Proceedings of the Eighteenth International Conference (ICML 2001), Stanford, California (2001)
17. Miyao, Y., Tsujii, J.: Maximum entropy estimation for feature forests. In: Proceedings of Human Language Technology Conference 2002. (2002)
18. Miyao, Y., Tsujii, J.: A model of syntactic disambiguation based on lexicalized grammars. In: Proceedings of the Seventh Conference on Natural Language Learning. (2003) 1–8
19. Smyth, P., Heckerman, D., Jordan, M.: Probabilistic Independence Networks for Hidden Markov Models. Neural Computation **9** (1997) 227–269
20. Pearl, J.: Probabalistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, California (1988)
21. Cowell, R.: Introduction to inference for Bayesian networks. In Jordan, M., ed.: Learning in Graphical Models. The MIT Press, Cambridge, Massachusetts (1999) 9–26
22. Geman, S., Kochanek, K.: Dynamic programming and the representation of soft-decodable codes. Technical report, Division of Applied Mathematics, Brown University (2000)