Maximum Entropy Estimation for Feature Forests

MIYAO Yusuke
Department of Computer Science,
University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 Japan
yusuke@is.s.u-tokyo.ac.jp

TSUJII Jun'ichi
Department of Computer Science,
University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 Japan
CREST, JST
(Japan Science and Technology Corporation)
tsujii@is.s.u-tokyo.ac.jp

ABSTRACT

An algorithm is proposed for maximum entropy modeling. It enables probabilistic modeling of complete structures, such as transition sequences in Markov models and parse trees, without dividing them into independent sub-events. A probabilistic event is represented by *a feature forest*, which is a packed representation of features with ambiguities. The parameters are efficiently estimated by traversing each node in a feature forest by dynamic programming. Experiments showed the algorithm worked efficiently even when ambiguities in a feature forest cause an exponential explosion of unpacked structures.

1. INTRODUCTION

Maximum entropy models [2] are widely used for probabilistic modeling of various tasks, such as part-of-speech tagging [9, 5] and parsing [10], because they achieve higher accuracy. However, studies so far have merely applied maximum entropy models to traditional, chain-rule-based models, such as n-grams and lexicalized versions of a probabilistic context-free grammar (PCFG). That is, an event is divided into *independent* sub-events, and the probability of the event is defined as the product of the probabilities of the sub-events. Such studies inherently restrict flexibility by independence assumption, which is not required for maximum entropy modeling.

The algorithm proposed in this paper enables probabilistic modeling of complete structures, such as transition sequences in Markov models and parse trees, without dividing them into *independent* sub-events. In general, complete structures have exponentially many possibilities, which is problematic in estimating parameters of a maximum entropy model. Our algorithm avoids an exponential explosion by representing an event by *a feature forest*, which is a packed representation of trees (or directed acyclic graphs). If a complete structure is represented by a feature forest of a tractable size, the parameters are efficiently estimated by dynamic programming similar to *the inside/outside algorithm* without unpacking the feature forest.

The proposed algorithm enables a more flexible model for various NLP tasks because it allows incorporating various overlapping features of a complete structure, not only of sub-events, into the model. Moreover, it can be applied to a probabilistic model of events that are difficult to divide into independent sub-events, such as a probabilistic model of feature structures [1].

Rosenfeld's study of a whole sentence maximum entropy model [11] had a similar motivation. However, the model was basically a sequence model, so it could not produce a solution for complex structures like our model can. Johnson et al. applied a maximum entropy model to a lexical functional grammar (LFG) [4], but they ignored the problem of an exponential explosion of unpacked parse results. We argue that the exponential explosion is inevitable, especially with large-scale wide-coverage grammars. In such cases, their method does not work. The forthcoming work of Johnson has the same motivation as ours [3]. The solution is similar to our approach, while their method is designed for directly traversing parse results with conditional disjunctions. We should also mention the work of Lafferty et al. in solving a similar problem in the context of maximum entropy Markov models [6]. Their solution was an algorithm similar to *the forward-backward algorithm*, and it is a special case of our algorithm in which each conjunctive node has only one daughter. This fact shows that our algorithm is applicable to various tasks, not only to parsing.

Section 2 overviews maximum entropy models and its applications. Section 3 discusses the problem with conventional maximum entropy modeling. Section 4 describes our algorithm for solving the problem. Section 5 discusses the evaluation of our algorithm using a parsing experiment.

2. MAXIMUM ENTROPY MODEL

Maximum entropy models [2] are widely used for probabilistic modeling of various NLP tasks, such as part-of-speech tagging [9, 5] and sentence parsing [10]. They do not require independence assumption to divide a probabilistic event into sub-events, enabling flexible modeling with many overlapping features. When modeling a probabilistic distribution of event $e = \langle t, h \rangle$, where t is *a target event* and h *a history event*, e is represented by a bundle of *feature functions* (or *features* for short) $f_i(e)$. A feature function represents the existence of a certain characteristic in event e, and a set of *activated* features, i.e., $f_i(e) \neq 0$, is considered to be an abstracted representation of an event.

Figure 1 shows an example of features in a simple maximum entropy bigram model for part-of-speech tagging[1]. As in conventional tagging models, an individual event is an assignment of a tag to a word, given previous words and tags as a context. The target event is a part-of-speech tag to be assigned, and the history event is

[1] Part-of-speech tags are of the Penn Treebank [7].

sentence	He	gave	books	to	his	sister .
part-of-speech	PRP	VBD	?			

feature	*target*	*current word*	*prev. word*	*prev. tag*
f_0	NNS	books	gave	-
f_1	VBZ	books	gave	-
f_2	NNS	books	gave	VBD
f_3	NNS	books	-	VBD
f_4	NNS	*ends_with_"s"*	-	-
f_5	VBZ	*ends_with_"s"*	-	-

Figure 1: Maximum entropy bigram model for part-of-speech tagging

a tuple $\langle$*current word, previous word, previous tag*$\rangle$. Each feature function represents a certain characteristic of the target and history events, and in particular, a maximum entropy model enables incorporating features that are not statistically independent, such as f_0, f_2 and f_3 in Figure 1. This enables flexible modeling with various kinds of overlapping features.

Formally, maximum entropy model p_M is a log-linear model that gives a conditional probability of event $e = \langle t, h\rangle$, where $\tau(h)$ is a set of targets observable with history h [2]:

$$p_M(t|h) = \frac{1}{Z_h}\prod_i \alpha_i^{f_i(t,h)}$$

$$\textit{where} \quad Z_h = \sum_{t'\in\tau(h)}\prod_i \alpha_i^{f_i(t',h)}$$

A feature function is an indicator for a certain characteristic of an event, and *model parameter* α_i corresponding to f_i is its weight. A maximum entropy model assigns a probability to an event by multiplying weights α_i when the corresponding features are activated, i.e., $f_i(e) \neq 0$. A maximum entropy model yields a probability distribution that maximizes the likelihood of the training data given a set of feature functions [2].

Note that the above model does not require independence of feature functions. It yields a maximum likelihood estimation even when the features are not independent, which is unattainable when using relative frequency estimation, as in traditional probabilistic models. This advantage enables flexible modeling with various kinds of overlapping feature functions, as described above. In addition, maximum entropy models can consistently model complex structures such as feature structures, which are difficult to decompose into independent events [1].

The improved iterative scaling (IIS) algorithm [8] (Figure 2) finds the model parameters that maximize the likelihood of the training data, $\tilde{p}(t,h)$, which is an observed distribution of events. The parameters are iteratively updated: $\alpha_i \leftarrow \alpha_i\Delta\alpha_i$, where $\Delta\alpha_i$ is the solution to the following equation.

$$\begin{aligned}\sum_{t,h}\tilde{p}(t,h)f_i(t,h) &= \sum_{t,h}\tilde{p}(h)p_M(t|h)f_i(t,h)\Delta\alpha_i^{f^\#(t,h)} \\ &= \sum_{f^\#}\Delta\alpha_i^{f^\#}\mu_{\langle i,f^\#\rangle}\end{aligned}$$

$$\textit{where} \quad f^\#(t,h) = \sum_i f_i(t,h)$$

The algorithm shown in Figure 2 implements this. Note that the algorithm is optimized by factoring the terms having the same

```
Input: training data p̃(t,h),
  feature functions f_i(t,h), initial parameters α_i
Output: optimal parameters α_i

E_i ← Σ p̃(t,h) f_i(t,h)
loop until α_i converges
  foreach e = ⟨t,h⟩
    foreach t' ∈ τ(h)
      foreach i such that f_i(t',h) ≠ 0
        μ_⟨i,f#(t',h)⟩ ← μ_⟨i,f#(t',h)⟩ + p̃(h) f_i(t',h) (1/Z_h) Π_k α_k^{f_k(t',h)}
      end_foreach
    end_foreach
  end_foreach
  foreach f_i
    Let Δα_i be the solution to the following equation:
      Σ_k μ_⟨i,k⟩ Δα_i^k = E_i
    α_i ← α_i Δα_i
  end_foreach
end_loop
```

Figure 2: Parameter estimation algorithm based on improved iterative scaling

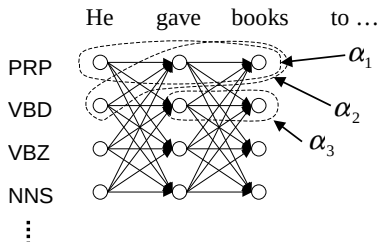


Figure 3: Maximum entropy tagging model without independence assumption

$f^\#(t,h)$. The main part of the algorithm is the computation of *a factored model expectation* $\mu_{\langle i,f^\#\rangle}$, which is the expectation of feature f_i given by the model.

$$\mu_{\langle i,f^\#\rangle} = \sum_h \tilde{p}(h)\sum_{t'\in\tau(h)} f_i(t',h)p_M(t'|h)$$

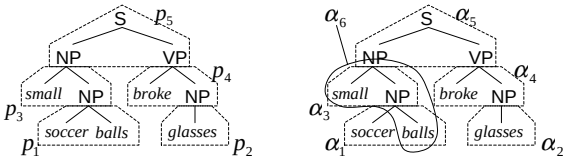
The computational complexity of each iteration is $O(|\mathcal{T}||\mathcal{E}||F|)$, where $|\mathcal{T}|$, $|\mathcal{E}|$, and $|F|$ are the number of targets and events, and the average number of activated features, respectively. The algorithm is sufficiently efficient for various applications, such as part-of-speech tagging, because $|\mathcal{T}|$ and $|F|$ are usually small (more or less than 100), while $|\mathcal{E}|$ is large, as in other statistical models.

However, this model/algorithm still cannot be applied in some cases due to the problem described in the next section.

3. PROBLEM

Let us consider the maximum entropy modeling of tagging tasks. Conventional models divide a tagging sequence into a tagging event for each word [9, 5]. That is, the models incorporate independence assumption, which is not required in maximum entropy models. Figure 3 shows a tagging model without independence assumption, where the complete tag sequence for a sentence is considered to be a target event. The model chooses the most probable tag sequence from the set of all possible tag sequences. We can incorporate various dependent features such as bigram/trigram/n-gram features as in the figure. This model is an extension of conventional models, and it has a more powerful modeling scheme [6].

A similar argument can be made for the maximum entropy modeling of parsing. In conventional parsing models, similar to PCFGs, each branching is assumed to be an independent event, and the



PCFG-like model
$p = p_1 p_2 p_3 p_4 p_5$

Maximum entropy model
$p = \frac{1}{Z}\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\cdots$

Figure 4: Probabilistic models for parsing

probability of the complete parse tree is defined as the product of the probabilities assigned to the branchings (left-hand side of Figure 4). On the other hand, we can consider a maximum entropy model that assigns a probability to the complete parse tree without independence assumption (right-hand side of Figure 4). Feature functions correspond to the branchings (or other characteristics such as α_6) in a parse tree. Given such features, the model is formulated as a probabilistic model of selecting parse tree t from a parse forest for sentence h. As described above, the maximum entropy model does not require independence assumption, and various overlapping features (not limited to immediate dominance relations) can be incorporated. Obviously, this model is a natural extension of PCFG, and has a more powerful modeling scheme.

The remaining issue we should consider is parameter estimation. The problem arises here: the number of targets t, (i.e., the number of possible tag sequences or parse trees for a sentence), is quite large in general. This is because local ambiguities in a tag sequence or a parse tree result in exponential growth in the number of structures, resulting in billions of structures. This is quite a problem, because, as described in Section 2, the complexity of parameter estimation is proportional to the number of targets. While a similar problem arises when using a PCFG to compute the probability of each parse tree, it can be avoided by using *the inside/outside algorithm*, which efficiently computes the probability of each parse tree by dynamic programming. Why don't we use the same approach?

This idea yields a new algorithm for parameter estimation, as described in the next section.

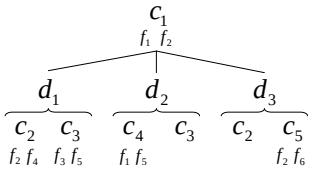
4. SOLUTION

Our solution to the problem is an efficient algorithm for parameter estimation using *the inside/outside α-product*. Similar to the inside/outside algorithm, we can compute the product of α_i for each node in tree structure. *An inside α-product* is the summation of each product of α_i in one of the daughter sub-trees. This value is incrementally computed by multiplying the inside α-products of the daughter nodes. *An outside α-product* is the summation of each product of α_i in the upper part of the trees. Given the inside α-products, this value is also incrementally computed by multiplying the outside α-product of the mother node and the inside α-products of the sister nodes. Given all inside/outside α-products, the model expectation, $\mu_{\langle i,f\#\rangle}$, is easily computed by multiplying them for each node.

To describe the algorithm, we first define *a feature forest*, a generalized representation of features in packed forest structure. Feature forests are used for enumerating possible structures of target events, that is, they correspond to $\tau(h)$ in conventional models described in Section 2.

DEFINITION 1 (FEATURE FOREST). *Feature forest Φ is a tuple $\langle C, D, r, \gamma, \delta\rangle$, where*

- *C is a set of conjunctive nodes*



$C = \{c_1, c_2, c_3, c_4, c_5\}$
$D = \{d_1, d_2, d_3\}$
$r = c_1$
$\gamma(d_1) = \{c_2, c_3\},\ \gamma(d_2) = \cdots$
$\delta(c_1) = \{d_1, d_2, d_3\},\ \delta(c_2) = \cdots$

Figure 5: A feature forest

- *D is a set of disjunctive nodes*
- *r is the root node: $r \in C$*
- *$\gamma : D \mapsto 2^C$ is a conjunctive daughter function*
- *$\delta : C \mapsto 2^D$ is a disjunctive daughter function*

Feature functions are defined over conjunctive nodes[2], and thus a feature forest represents a set of trees of features, which are associated to conjunctive nodes. We denote a feature forest for history h (for a CFG, the activated features of a parse forest for a sentence) as $\Phi(h)$. We assume that a feature forest is acyclic and can be a directed acyclic graph. It is evident that the models introduced in Section 3 can be represented with this structure.

Figure 5 shows an example feature forest. Each disjunctive node has alternative nodes, which are conjunctive nodes. Each conjunctive node has activated features, i.e., $f_i(c) \neq 0$; it also has disjunctive nodes as daughters. A feature forest can include a reentrant structure, that is, a node can appear more than once in a feature forest, such as c_3 in d_1 and d_2.

Since Figure 5 resembles a parse forest of a PCFG, it might be asserted that a feature forest can represent only immediate dominance relations in each node, as in a PCFG, resulting in only a slight, trivial extension of the PCFG. However, a feature forest is a generalized representation of an ambiguous structure, and each node in a feature forest does not need to correspond to a node in a parse forest. That is, a node in a feature forest can represent any linguistic entity, including a fragment of a syntactic/semantic structure and other sentence-level information. Instances of a node can thus be neglected in the following discussion, and our algorithm is applicable whenever the feature forest is of a tractable size. However, a feature forest might be sometimes too large due to incorporating sentence-level features. In those cases, an approximation method to reduce the computational cost might be required [11].

As mentioned, a feature forest is a packed representation of feature sets, and each feature set is extracted by using the following function.

DEFINITION 2 (UNPACKING). Unpacking function *σ is a function*

- *σ: $C \mapsto 2^C$*

such that

$$\sigma(c) = \bigcup_{d_i \in \delta(c)} \{c_i\} \quad \textit{where} \quad c_i \in \gamma(d_i)$$

[2]While feature functions can also depend on history events, we omit the histories in the following discussion for simplicity.

We extend function σ to take set C as an argument:

$$\sigma(C) = \bigcup_{c \in C} \sigma(c)$$

Intuitively, unpacking function σ selects one conjunctive node for each disjunctive node. By applying this function to each node, we can extract the set of conjunctive nodes. Let us denote multiple applications of σ as σ^n, and a tree extracted from a feature forest is defined as follows.

DEFINITION 3 (UNPACKED TREE). *An unpacked tree rooted at conjunctive node $c \in C$ is defined as*

$$\sigma^*(c) = \bigcup_i \sigma^i(\{c\})$$

Each unpacked tree is then represented by $\sigma^*(r)$, and each function σ corresponds to each unpacked tree. Note that an unpacked tree is equivalent to a set of conjunctive nodes because our concern is only the set of features associated to each conjunctive node; the shape of the tree structure is irrelevant to probabilistic models. In this paper, we do not distinguish an unpacked tree from a set of conjunctive nodes.

The feature forest in Figure 5 represents a set of $2 \times 2 \times 2 = 8$ unpacked trees. For example, if we select the left-most conjunctive node at each disjunctive node, we extract tree $\{c_1, c_2, c_4, c_2\}$. Generally, a feature forest can represent an exponential number of trees with a polynomial number of nodes. Thus, a complete structure, such as tag sequences and parse trees with ambiguities, can be represented compactly.

Given the above formalization, a feature function for an unpacked tree is

$$f_i(\sigma^*(r)) = \sum_{c \in \sigma^*(r)} f_i(c)$$

Once a feature function for an unpacked tree is given, a factored model expectation can be defined as in the traditional model described in Section 2.

DEFINITION 4 (MODEL EXPECTATION). *Factored model expectation μ for a feature forest is defined as*

$$\mu_{\langle i, f^\# \rangle} = \sum_h \tilde{p}(h) \frac{1}{Z_h} \sum_\sigma f_i(\sigma^*(r)) \prod_{c \in \sigma^*(r)} \prod_{f_j} \alpha_j^{f_j(c)}$$

It is evident that the naive computation of factored model expectations requires exponential time complexity since the number of unpacked trees (i.e., σ) is exponentially related to the number of nodes in the feature forest. We thus need an algorithm for computing factored model expectations without unpacking a feature forest.

To efficiently compute the factored model expectations, we incorporate an approach similar to the inside/outside algorithm of a PCFG. We first define the notion of inside/outside for a feature forest.

DEFINITION 5 (INSIDE/OUTSIDE). *We define the* inside, $\iota(c)$, *of a feature forest at conjunctive node $c \in C$ as a set of unpacked trees:*

$$\iota(c) = \{C_i | C_i = \sigma_i^*(c)\}$$

The outside, $o(c)$, *is defined as*

$$o(c) = \{C_i \backslash \sigma_i^*(c) | C_i = \sigma_i^*(r)\}$$

Figure 6 illustrates this concept, which is similar to that of a PCFG. Inside denotes a set of partial trees (conjunctive nodes) derived from node c. Outside denotes the complement of the inside trees.

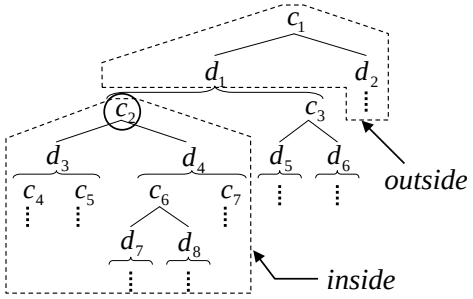


Figure 6: Inside/outside at node c_5 in a feature forest

An inside/outside α-product is then defined for each conjunctive/disjunctive node. The inside/outside α-products are the summation of the products of α in the inside/outside trees.

DEFINITION 6 (INSIDE/OUTSIDE α-PRODUCT). *An inside α-product at conjunctive node $c \in C$ is*

$$\phi_{\langle c, f^\# \rangle} = \sum_{C' \in \iota(c)} \prod_{c' \in C'} \prod_{f_i} \alpha_i^{f_i(c')}$$

$$\textit{such that} \quad f^\# = \sum_{c' \in C'} \sum_{f_i} f_i(c')$$

An outside α-product is

$$\psi_{\langle c, f^\# \rangle} = \sum_{C' \in o(c)} \prod_{c' \in C'} \prod_{f_i} \alpha_i^{f_i(c')}$$

$$\textit{such that} \quad f^\# = \sum_{c' \in C'} \sum_{f_i} f_i(c')$$

We can also define inside/outside α-products for disjunctive nodes:

$$\phi_{\langle d, f^\# \rangle} = \sum_{c \in \gamma(d)} \phi_{\langle c, f^\# \rangle}$$

$$\psi_{\langle d, f^\# \rangle} = \sum_{\{c | d \in \delta(c)\}} \psi_{\langle c, f^\# \rangle}$$

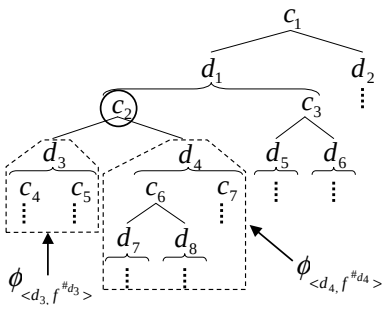
We can derive that the factored model expectation of a feature forest is defined as the product of the inside and outside α-products.

$$\mu_{\langle i, f^{\#_\phi} + f^{\#_\psi} \rangle} = \sum_h \tilde{p}(h) \frac{1}{Z_h} \sum_{c \in C} f_i(c) \phi_{\langle c, f^{\#_\phi} \rangle} \psi_{\langle c, f^{\#_\psi} \rangle}$$

This equation shows a method for efficiently computing the factored model expectation by traversing conjunctive nodes without unpacking the forest, if the inside/outside α-products are given. The remaining issue is how to efficiently compute the inside/outside α-products.

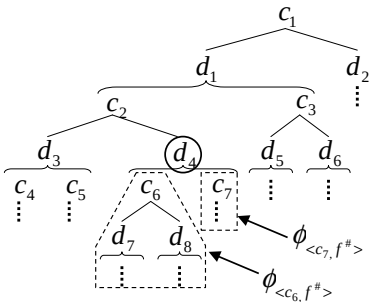
Fortunately, the inside/outside α-products can be incrementally computed by dynamic programming without unpacking the feature forest. Figure 7 shows the process of computing an inside α-product at a conjunctive node from the inside α-products of its daughter nodes. Since an inside α-product is the product of all α_i of its descendants, it is computed by multiplying the α-products of the daughter trees. We should take care of the computation of an index $f^\#$, and the following equation is derived.

$$\phi_{\langle c, f^\# \rangle} = \prod_{d \in \delta(c)} \phi_{\langle d, f^{\#_d} \rangle} \prod_{f_i} \alpha_i^{f_i(c)}$$



$$\phi_{\langle c_2, f^{\#d_3}+f^{\#d_4}+f^{\#}\rangle} = \phi_{\langle d_3, f^{\#d_3}\rangle}\phi_{\langle d_4, f^{\#d_4}\rangle}\prod_{f_i}\alpha_i^{f_i(c_2)}$$

Figure 7: Incremental computation of inside α-products at conjunctive node c_2



$$\phi_{\langle d_4, f^{\#}\rangle} = \phi_{\langle c_6, f^{\#}\rangle} + \phi_{\langle c_7, f^{\#}\rangle}$$

Figure 8: Incremental computation of inside α-products at disjunctive node d_4

$$\text{such that} \quad f^{\#} = \sum_d f^{\#d} + \sum_i f_i$$

The inside of a disjunctive node is the collection of the inside trees of its daughter nodes. Hence, the inside α-product at disjunctive node $d \in D$ is computed as follows (Figure 8).

$$\phi_{\langle d, f^{\#}\rangle} = \sum_{c \in \gamma(d)} \phi_{\langle c, f^{\#}\rangle}$$

The outside of a disjunctive node is equivalent to the outside of its daughter nodes, and the outside α-product of a disjunctive node is propagated to its daughter conjunctive nodes (Figure 9).

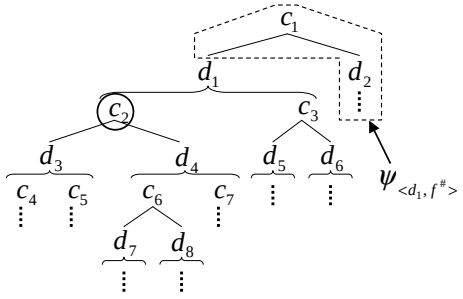
$$\psi_{\langle c, f^{\#}\rangle} = \sum_{\{d | c \in \gamma(d)\}} \psi_{\langle d, f^{\#}\rangle}$$

The computation of an outside α-product of a disjunctive node is a little more complicated. The outside trees of a disjunctive node are all combinations of (Figure 10)

- the outside trees of the mother nodes and
- the inside trees of the sister nodes.

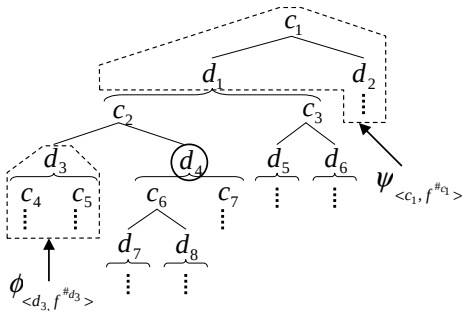
From this, we get

$$\psi_{\langle d, f^{\#}\rangle} = \sum_{\{c | d \in \delta(c)\}} \left\{ \psi_{\langle c, f^{\#c}\rangle} \prod_{\{d' | d' \in \delta(c), d' \neq d\}} \phi_{\langle d', f^{\#d'}\rangle} \right\}$$



$$\psi_{\langle c_2, f^{\#}\rangle} = \psi_{\langle d_1, f^{\#}\rangle}$$

Figure 9: Incremental computation of outside α-products at conjunctive node c_2



$$\psi_{\langle d_4, f^{\#c_2}+f^{\#d_3}\rangle} = \psi_{\langle c_2, f^{\#c_2}\rangle}\phi_{\langle d_3, f^{\#d_3}\rangle}$$

Figure 10: Incremental computation of outside α-products at disjunctive node d_4

$$\text{such that} \quad f^{\#} = f^{\#c} + \sum_{d'} f^{\#d'}$$

Figure 11 shows the overall algorithm for estimating the parameters, given a set of feature forests. The key point of the algorithm is to compute inside α-products ϕ and outside α-products ψ for each node in C, not for all unpacked trees. The functions `inside_product` and `outside_product` compute ϕ and ψ efficiently by dynamic programming. Note that the order of traversing nodes is important for incremental computation, although it is not shown in Figure 11. The computation for the daughter nodes and mother nodes must be finished before computing the inside and outside α-products, respectively. This constraint is easily solved using a topological sort.

The complexity of this algorithm is $O((|C|+|D|)|\mathcal{E}||F|)$, which is tractable when $|C|$ and $|D|$ are of a reasonable size. As noted in this section, the number of nodes in a feature forest is usually a polynomial even when that of the unpacked trees is exponential. This result shows that we can efficiently compute factored model expectations with polynomial computational complexity.

5. EVALUATION

The proposed algorithm was implemented in a maximum entropy estimator, and its correctness was tested experimentally by comparing the output of the original IIS with that of our algorithm. We also evaluated the efficiency and applicability of the algorithm by using a probabilistic model of an automatically extracted lexicalized tree adjoining grammar (LTAG).

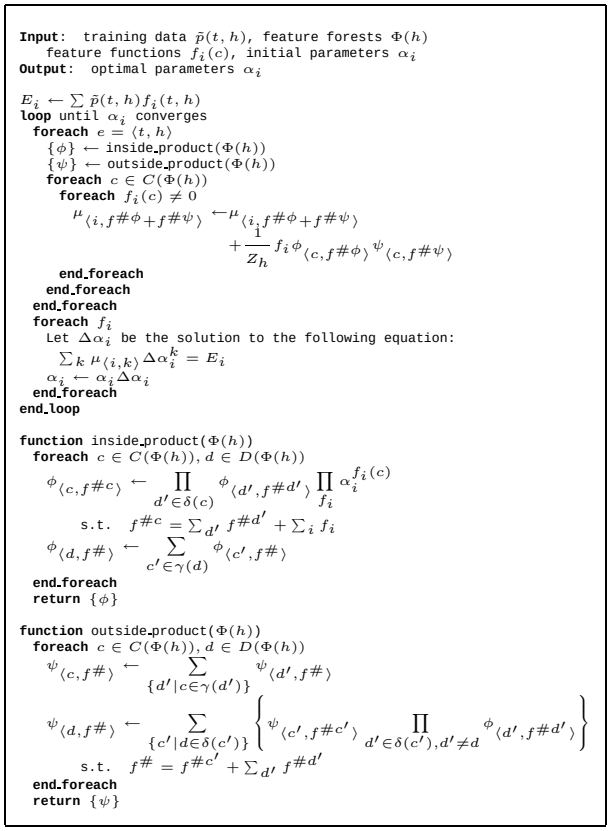


Figure 11: Parameter estimation algorithm for feature forests

Table 1: Experimental results

number of features	5715
number of sentences	868
avg. number of nodes	17412
required memory	1532 MB
required time	547 min.

The LTAG grammar we used was extracted from Section 00 of the Penn Treebank [7]. This section consists of 1603 sentences. The grammar was consistently extracted from about a half of the sentences (868 sentences), and consisted of 826 elementary tree templates for 3275 words. We parsed the sentences used for the grammar extraction and extracted the feature forests from the parse results. The parse trees were lexicalized; that is, each node in each tree was lexicalized using its head word. Each lexicalized parse forest consisted of 17,412 nodes on average; the number of unpacked parse trees could not be counted because it reached the limit for the `int` type (i.e., $> 2^{32}$).

Table 1 shows the results of estimating parameters using 100 iterations on a Pentium III 550-MHz CPU. In spite of the exponential growth in the number of unpacked parse trees, the estimator worked with a tractable cost. The results also show the algorithm costs somewhat more than traditional maximum entropy models. This is mainly because $|C| + |D|$ is much larger (17,412 on average, and more than 1,000,000 at the maximum) than with conventional models (100 or so). Although the proposed algorithm is more costly than conventional ones, it provides more flexibility and applicability with a tractable cost.

6. CONCLUSION

A new algorithm was presented for maximum entropy modeling and shown to be applicable to probabilistic parsing. Experimental results showed that it enables parameter estimation of probabilistic models of complete structures without independence assumption. It provides a more flexible modeling scheme than conventional algorithms, and furthermore, it is applicable to more complex structures where an event is difficult to decompose into independent subevents, such as probabilistic modeling of feature structures.

Future work includes precise comparison of parsing models based on our algorithm with conventional ones. Furthermore, the model should be applicable to more complex representations, such as derivation trees in LTAG, feature structures, and predicate-argument structures. The algorithm described in this paper provides a consistent solution for probabilistic modeling of complex structures.

7. ACKNOWLEDGMENTS

We are very grateful to Dr. Takashi Ninomiya for his useful suggestions and encouragement. We would also like to thank the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] S. P. Abney. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4), 1997.

[2] A. L. Berger, S. A. D. Pietra, and V. J. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[3] M. Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. To appear in Proc. 40th ACL, 2002.

[4] M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler. Estimators for stochastic "unification-based" grammars. In *Proc. ACL '99*, pages 535–541, 1999.

[5] J. Kazama, Y. Miyao, and J. Tsujii. A maximum entropy tagger with unsupervised hidden Markov models. In *Proc. NLPRS2001*, 2001.

[6] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning 2001*, 2001.

[7] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn Treebank: Annotating predicate argument structure. In *Proc. AAI '94*, 1994.

[8] S. D. Pietra, V. D. Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[9] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proc. Empirical Methods in Natural Language Processing Conference*, 1996.

[10] A. Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proc. Empirical Methods in Natural Language Processing Conference*, 1997.

[11] R. Rosenfeld. A whole sentence maximum entropy language model. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997.