Forest-Based Statistical Sentence Generation

Irene Langkilde
Information Sciences Institute
University of Southern California
Marina del Rey CA 90292
ilangkil@isi.edu

Abstract

This paper presents a new approach to statistical sentence generation in which alternative phrases are represented as packed sets of trees, or forests, and then ranked statistically to choose the best one. This representation offers advantages in compactness and in the ability to represent syntactic information. It also facilitates more efficient statistical ranking than a previous approach to statistical generation. An efficient ranking algorithm is described, together with experimental results showing significant improvements over simple enumeration or a lattice-based approach.

1 Introduction

Large textual corpora offer the possibility of a statistical approach to the task of sentence generation. Like any large-scale NLP or AI task, the task of sentence generation requires immense amounts of knowledge. The knowledge needed includes lexicons, grammars, ontologies, collocation lists, and morphological tables. Acquiring and applying accurate, detailed knowledge of this breadth poses difficult problems.

Knight and Hatzivassiloglou (1995) suggested overcoming the knowledge acquisition bottleneck in generation by tapping the information inherent in textual corpora. They performed experiments showing that automatically-acquired, corpus-based knowledge greatly reduced the need for deep, hand-crafted knowledge. At the same time, this approach to generation improved scalability and robustness, offering the potential in the future for higher quality output.

In their approach, K & H adapted techniques used in speech recognition. Corpus-based statistical knowledge was applied to the generation process after encoding many alternative phrasings into a structure called a *lattice* (see Figure 1). A lattice was able to represent large numbers alternative phrases without requiring the large amount of space that an explicitly enumerated list of individual alternatives would require. The alternative sentences in the lattice were then ranked according to a statistical language model, and the most likely sentence was chosen as output. Since the number of phrases that needed be considered typically grew exponentially with the length of the phrase, the lattice was usually too large for an exhaustive search, and instead an n-best algorithm was used to heuristically narrow the search.

The lattice-based method, though promising, had several drawbacks that will be discussed shortly. This paper presents a different method of statistical generation based on a *forest* structure (a packed set of trees). A forest is more compact than a lattice, and it offers a hierarchical organization that is conducive to representing syntactic information. Furthermore, it facilitates dramatically more efficient statistical ranking, since constraints can be localized, and the combinatorial explosion of possibilities that need be considered can be reduced. In addition to describing the forest data structure we use, this paper presents a forest-based ranking algorithm, and reports experimental results on its efficiency in both time and space. It also favorably compares these results to the performance of a lattice-based approach.

2 Representing Alternative Phrases

2.1 Enumerated lists and lattices

The task of sentence generation involves mapping from an abstract representation of meaning or syntax to a linear ordering of words. Subtasks of generation usually include choosing content words, determining word order,

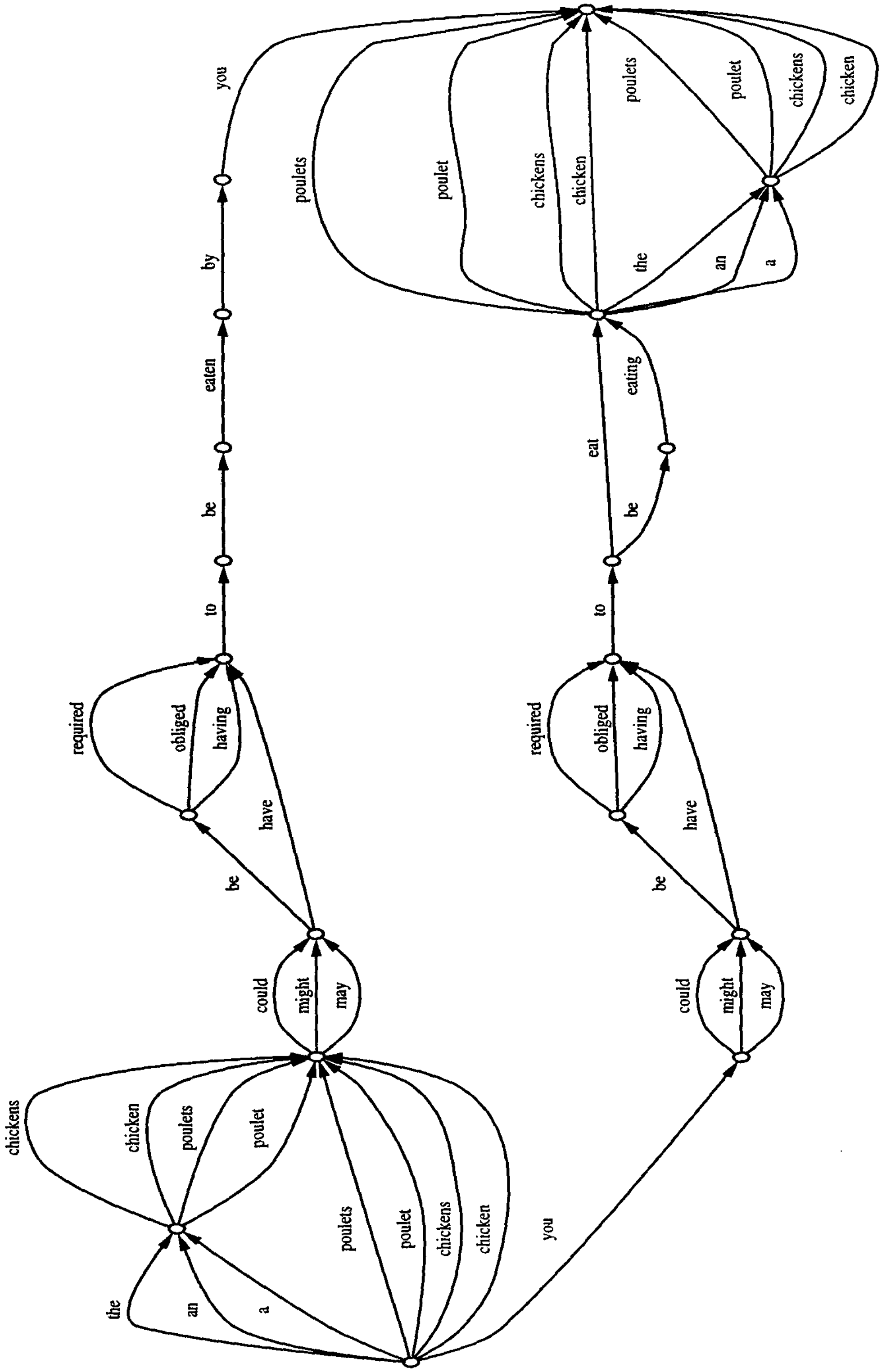


Figure 1: A lattice representing 576 different sentences, including "You may have to eat chicken", "The chicken may have to be eaten by you", etc.

deciding when to insert function words, performing morphological inflections, and satisfying agreement constraints, as well as other tasks.

One way of leveraging corpus-based knowledge is to explicitly enumerate many alternate possibilities and select the most likely according to a corpus-based statistical model. Since many subphrases and decisions will be common across proposed sentences, a lattice is a more efficient way than one-by-one enumeration to represent them. A lattice is a graph where each arc is labeled with a word. A complete path from the left-most node to right-most node through the lattice represents a possible sentence. Multiple arcs leaving a particular node represent alternate paths. A lattice thus allows structure to be shared between sentences. An example of a lattice is shown in Figure 1. This lattice encodes 576 unique sentences. In practice, a lattice may represent many trillions of sentences. Without a compact representation for so many sentences, statistical generation would be much less feasible.

The lattice in Figure 1 illustrates several types of decisions that need to be made in generation. For example, there is a choice between the root words "chicken" and "poulet", the choice of whether to use singular or plural forms of these words, the decision whether to use an article or not, and if so, which one—definite or indefinite. There are also other word choice decisions such as whether to use the auxiliary verb "could", "might", or "may", and whether to express the mode of eating with the predicate "have to", "be obliged to", or "be required to". Finally, there is a choice between active voice (bottom half of lattice), and passive voice (top half).

Inspection of the lattice reveals some unavoidable duplication, however. For example, the word "chicken" occurs four times, while the sublattice for the noun phrase containing "chicken" is repeated twice. So is the verb phrase headed by the auxiliaries "could", "might", and "may". Such repetition is common in a lattice representation for text generation, and has a negative impact on the efficiency of the ranking algorithm because the same set of score calculations end up being made several times. Another drawback of the duplication is that the representation consumes more storage space than necessary.

Yet another drawback of the lattice representation is that the independence between many choices cannot be fully exploited. Stolcke et al. (1997) noted that 55% of all word dependencies occur between adjacent words. This means that most choices that must be made in non-adjacent parts of a sentence are independent. For example, in Figure 1, the choice between "may", "might", or "could" is independent of the choice between "a", "an" or "the" to precede "chicken" or "poulet". Independence reduces the combination of possibilities that must be considered, and allows some decisions to be made without taking into account the rest of the context. Even adjacent words are sometimes independent of each other, such as the words "tail" and "ate" in the sentence "The dog with the short tail ate the bone". A lattice does not offer any way of representing which parts of a sentence are independent of each other, and thus cannot take advantage of this independence. This negatively impacts both the amount of processing needed and the quality of the results. In contrast, a forest representation, which we will discuss shortly, does allow the independence to be explicitly annotated.

A final difficulty with using lattices is that the search space grows exponentially with the length of the sentence(s), making an exhaustive search for the most likely sentence impractical for long sentences. Heuristic-based searches offer only a poor approximation. Any pruning that is done renders the solution theoretically inadmissable, and in practice, frequently ends up pruning the mathematically optimal solution.

2.2 Forests

These weaknesses of the lattice representation can be overcome with a forest representation. If we assign a label to each unique arc and to each group of arcs that occurs more than once in a lattice, a lattice becomes a forest, and the problems with duplication in a lattice are eliminated. The resulting structure can be represented as a set of context-free rewrite rules. Such a forest need not necessarily comply with a particular theory of syntactic structure, but it can if one wishes. It also need not be derived specifically from a lattice, but can be generated directly from a semantic input.

With a forest representation, it is quite natural to incorporate syntactic information. Syntactic information offers some potentially significant advantages for statistical language modeling. However, this paper will not discuss statistical modeling of syntax beyond making mention of it, leaving it instead for future work. Instead we focus on the nature of the forest representation itself and describe a general algorithm for ranking alternative trees that can be used with any language model.

A forest representation corresponding to the lattice in Figure 1 is shown in Figure 3. This forest structure is an AND-OR graph, where the AND nodes represent sequences of phrases, and the OR nodes represent mutually exclusive alternate phrasings for a particular relative position in the sentence. For example, at the top level of the forest, node S.469 encodes the choice between active and passive voice versions of the sentence. The active voice version is the left child node, labelled S.328, and the passive voice version is the right child node, S.358. There are eight OR-nodes in the forest, corresponding to the eight distinct decisions mentioned earlier that need to be made in deciding the best sentence to output.

The nodes are uniquely numbered, so that repeated references to the same node can be identified as such. In the forest diagram, only the first (left-most) reference to a node is drawn completely. Subsequent references only show the node name written in italics. This eases readability and clarifies which portions of the forest actually need to have scores computed during the ranking process. Nodes N.275, NP.318, VP.225 and PRP.3 are repeated in the forest of Figure 3.

S.469	⟹	S.328
S.469	⟹	S.358
S.328	⟹	PRP.3 VP.327
PRP.3	⟹	"you"
VP.327	⟹	VP.248 NP.318
S.358	⟹	NP.318 VP.357
NP.318	⟹	NP.317
NP.318	⟹	N.275

Figure 2: Internal representation of top nodes in forest

Figure 2 illustrates how the forest is represented internally, showing context-free rewrite rules for some of the top nodes in the forest. OR-nodes are indicated by the same label occuring more than once on the left-hand side of a rule. This sample of rules includes an example of multiple references to a node, namely node NP.318, which occurs on the right-hand side of two different rules.

A generation forest differs from a parse forest in that a parse forest represents different possible hierarchical structures that cover a single phrase. Meanwhile a generation forest generally represents one (or only a few) heirarchical structures for a given phrase, but represents many different phrases that generally express the same meaning.

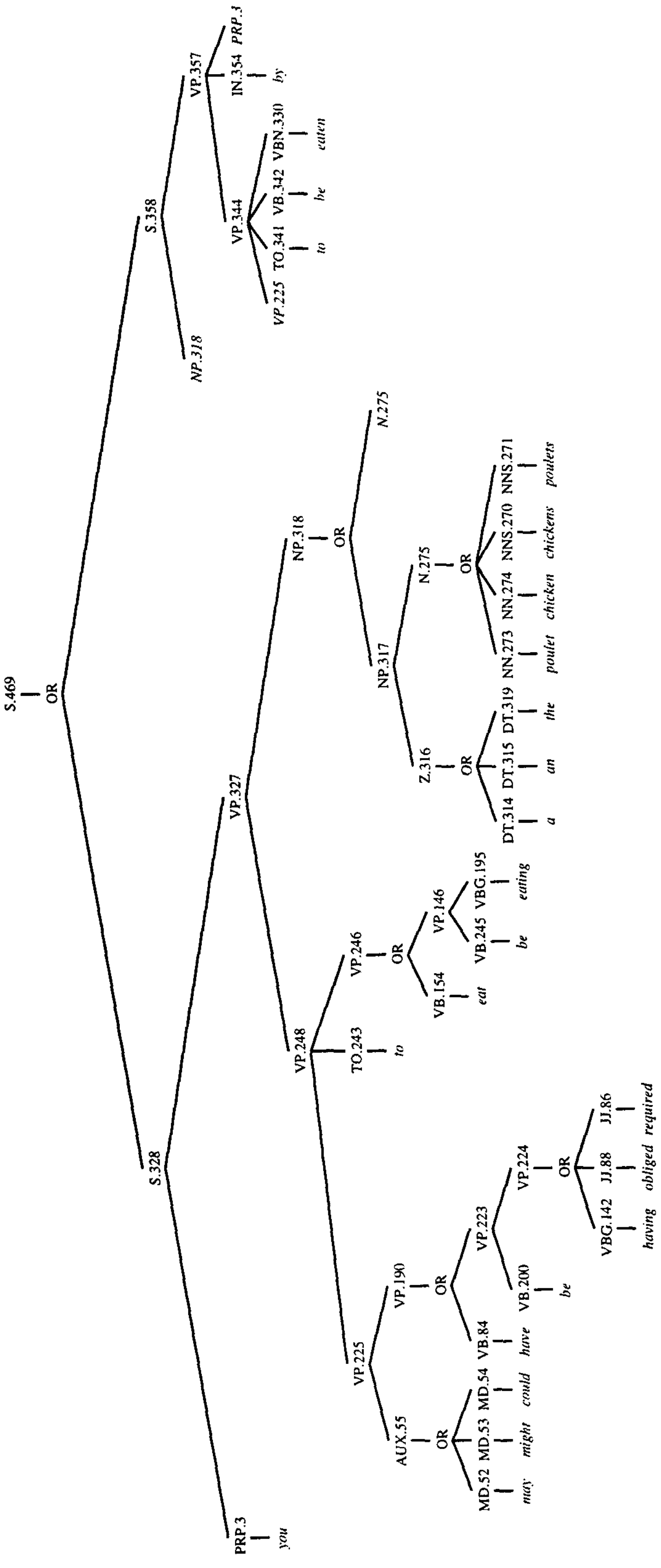
2.3 Previous work on packed generation trees

There has been previous work on developing a representation for a packed generation forest structure. Shemtov (1996) describes extensions to a chart structure for generation originally presented in (Kay, 1996) that is used to generate multiple paraphrases from a semantic input. A prominent aspect of the representation is the use of boolean vector expressions to associate each sub-forest with the portions of the input that it covers and to control the unification-based generation process. A primary goal of the representation is to guarantee that each part of the semantic input is expressed once and only once in each possible output phrase.

In contrast, the packed forest in this paper keeps the association between the semantic input and nodes in the forest separate from the forest representation itself. (In our system, these mappings are maintained via an external cache mechanism as described in (Langkilde and Knight, 1998)). Once-and-only-once coverage of the semantic input is implicit, and is achieved by the process that maps from the input to a forest.

3 Forest ranking algorithm

The algorithm proposed here for ranking sentences in a forest is a bottom-up dynamic programming algorithm. It is analogous to a chart parser, but performs an inverse comparison. Rather than comparing alternate syntactic structures indexed to the same positions of an



Comments: *italicized node numbers indicate duplicate nodes*

Figure 3: A generation forest

input sentence, it compares alternate phrases corresponding to the same semantic input.

As in a probabilistic chart parser, the key insight of this algorithm is that the score for each of the phrases represented by a particular node in the forest can be decomposed into a context-independent (internal) score, and a context-dependent (external) score. The internal score, once computed, is stored with the phrase, while the external score is computed in combination with other sibling nodes.

In general, the internal score for a phrase associated with a node p can be defined recursively as:

$$I(p) = \prod_{j=1}^{J} I(c_j) * E(c_j|context(c_1..c_{j-1}))$$

where I stands for the internal score, E the external score, and c_j for a child node of p. The specific formulation of I and E, and the precise definition of the *context* depends on the language model being used. As an example, in a bigram model,[1] I=1 for leaf nodes, and E can be expressed as:

$$E = P(FirstWord(c_j)|LastWord(c_{j-1}))$$

Depending on the language model being used, a phrase will have a set of externally-relevant *features*. These features are the aspects of the phrase that contribute to the context-dependent scores of sibling phrases. In the case of the bigram model, the features are the first and last words of the phrase. In a trigram model it is the first and last *two* words. In more elaborate language models, features might include elements such as head word, part-of-speech tag, constitutent category, etc.

A crucial advantage of the forest-based method is that at each node only the best internally scoring phrase for each unique combination of externally relevant features needs to be maintained. The rest can be pruned without sacrificing the guarantee of obtaining the overall optimal solution. This pruning reduces exponentially the total number of phrases that need to be considered. In effect, the ranking algorithm exploits the independence that exists between most disjunctions in the forest.

VP.344 $\Longrightarrow$ VP.225 TO.341 VB.342 VBN.330

225:	341:	342:	330:
might have	to	be	eaten
may have			
could have			
might be required			
may be required			
could be required			
might be having			
may be having			
could be having			
might be obliged			
may be obliged			
could be obliged			

344:
might ... eaten
may ... eaten
could ... eaten

Figure 4: Pruning phrases from a forest node, assuming a bigram model

To illustrate this, Figure 4 shows an example of how phrases in a node are pruned, assuming a bigram model. The rule for node VP.344 in the forest of Figure 3 is shown, together with the phrases corresponding to each of the child nodes. If every possible combination of phrases is considered for the sequence of nodes on the right-hand side, there are three unique first words, namely "might", "may" and "could", and only one unique final word, "eaten". Given that only the first and last words of a phrase are externally relevant features in a bigram model, only the three best scoring phrases (out of the 12 total) need to be maintained for node VP.344—one for each unique first-word and last-word pair. The other nine phrases can never be ranked higher, no matter what constituents VP.344 later combines with.

Pseudocode for the ranking algorithm is shown below. "Node" is assumed to be a record composed at least of an array of child nodes, "Node->c[1..N]," and best-ranked phrases, "Node->p[1..M]." The function ConcatAndScore concatenates two strings together, and computes a new score for it based on the formula given above. The function Prune guar-

[1] A bigram model is based on conditional probabilities, where the likelihood of each word in a phrase is assumed to depend on only the immediately previous word. The likelihood of a whole phrase is the product of the conditional probabilities of each of the words in the phrase.

antees that only the best phrase for each unique set of features values is maintained. The core loop in the algorithm considers the children of the node one-by-one, concatenating and scoring the phrases of the first two children and pruning the results, before considering the phrases of the third child, and concatenating them with the intermediate results, and so on. From the pseudocode, it can be seen that the complexity of the algorithm is dominated by the number of phrases associated with a node (not the number of rules used to represent the forest, nor the number of children in a an AND node). More specifically, because of the pruning, it depends on the number of features associated with the language model, and the average number of unique combinations of feature values that are seen. If f is the number of features, v the average number of unique values seen in a node for each feature, and N the number of N best being maintained for each unique set of feature values (but not a cap on the number of phrases), then the algorithm has the complexity $O((vN)^{2f})$ (assuming that children of AND nodes are concatenated in pairs). Note that f=2 for the bigram model, and f=4 for the trigram model.

In comparison, the complexity of an exhaustive search algorithm on a lattice is $O((vN)^l)$, where l is approximately the length of the longest sentence in the lattice. The forest-based algorithm thus offers an exponential reduction in complexity while still guaranteeing an optimal solution. A capped N-best heuristic search algorithm on the other hand has complexity O(vNl). However, as mentioned earlier, it typically fails to find the optimal solution with longer sentences.

```
RankForest( Node)
{
  if ( Leafp( Node)) LeafScore( Node);
  for j=1 to J {
      if ( not( ranked?(Node->c[j])))
          RankForest(Node->c[j]);
  }
  for m=1 to NumberOfPhrasesIn( Node->c[1])
      Node->p[m] = (Node->c[1])->p[m];
  k=0;for j=2 to J {
      for m=1 to NumberOfPhrasesIn( Node)
          for n=1 to NumberOfPhrasesIn(
                      Node->c[j])
              temp[k++] = ConcatAndScore(
                      Node->p[m],
                      (Node->c[j])->p[n]);
      Prune( temp);
      for m=1 to NumberOfPhrasesIn( temp)
          Node->p[m] = (temp[m]);
  }
}
```

In conclusion, the tables in Figure 5 and Figure 6 show experimental results comparing a forest representation to a lattice in terms of the time and space used to rank sentences. These results were generated from 15 test set inputs, whose average sentence length ranged from 14 to 36 words. They were ranked using a bigram model. The experiments were run on a Sparc Ultra 2 machine. Note that the time results for the lattice are not quite directly comparable to those for a forest because they include overhead costs for loading portions of a hash table. It was not possible to obtain timing measurements for the search algorithm alone. We estimate that roughly 80% of the time used in processing the lattice was used for search alone. Instead, the results in Figure 5 should be interpreted as a comparison between different kinds of systems.

In that respect, it can be observed from Table 5 that the forest ranking program performs at least 3 or 4 seconds faster, and that the time needed does *not* grow linearly with the number of paths being considered as it does with the lattice program. Instead it remains fairly constant. This is consistent with the theoretical result that the forest-based algorithm does not depend on sentence length, but only on the number of different alternatives being considered at each position in the sentence.

From Table 6 it can be observed that when there are a relatively moderate number of sentences being ranked, the forest and the lattice are fairly comparable in their space consumption. The forest has a little extra overhead in representing hierarchical structure. However, the space requirements of a forest do not grow linearly with the number of paths, as do those of the lattice. Thus, with very large numbers of paths, the forest offers significant savings in space.

The spike in the graphs deserves particular comment. Our current system for producing forests from semantic inputs generally produces OR-nodes with about two branches. The particular input that triggered the spike produced a forest where some high-level OR-nodes had a much larger number of branches. In a lattice, any increase in the number of branches exponentially increases the processing time and storage space requirements. However, in the forest representation, the increase is only polynomial with the number of branches, and thus did not produce a spike.

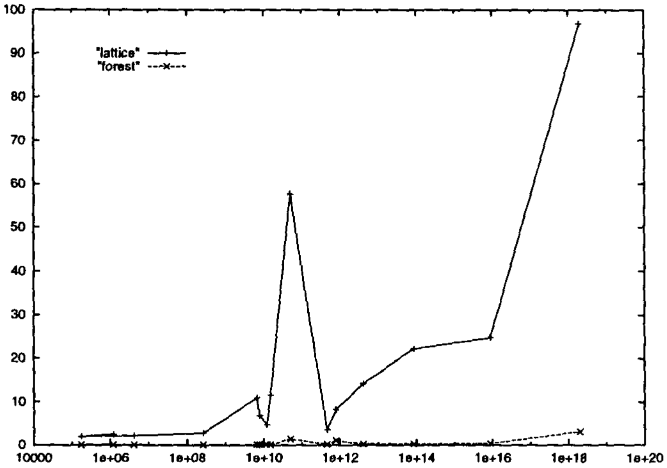


Figure 5: Time required for the ranking process using a lattice versus a forest representation. The X-axis is the number of paths (log_{10} scale), and the Y-axis is the time in seconds.

4 Future Work

The forest representation and ranking algorithm have been implemented as part of the Nitrogen generator system. The results shown in the previous section illustrate the time and space advantages of the forest representation which make calculating the mathematically optimal sentence in the forest feasible (particularly for longer sentences). However, obtaining the mathematically optimal sentence is only valuable if the mathematical model itself provides a good fit. Since a forest representation makes it possible to add syntactic information to the mathematical model, the next question to ask is whether such a model can provide a better fit for natural English than the ngram models we have used previously. In future work, we plan to modify the forests our system produces so they conform to the Penn Treebank corpus (Marcus et al., 1993) annotation style, and then do experiments using models built with Treebank data.

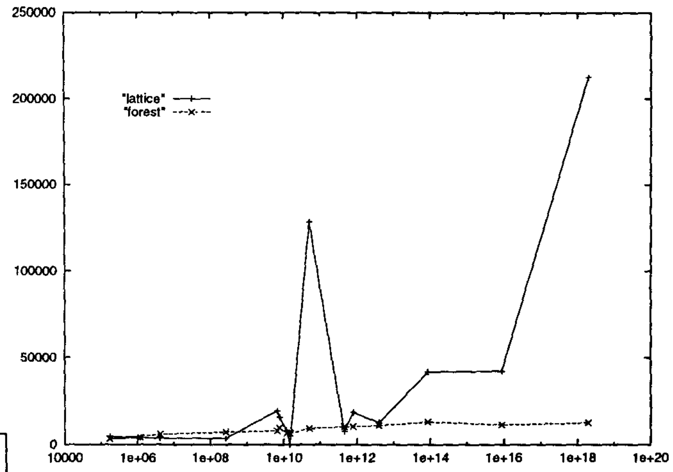


Figure 6: Size of the data structure for a lattice versus a forest representation. The X-axis is the number of paths (log_{10} scale), and the Y-axis is the size in bytes.

5 Acknowledgments

Special thanks go to Kevin Knight, Daniel Marcu, and the anonymous reviewers for their comments. This research was supported in part by NSF Award 9820291.

References

M. Kay. 1996. Chart generation. In *Proc. ACL.*

K. Knight and V. Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proc. ACL.*

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proc. COLING-ACL.*

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2).

H. Shemtov. 1996. Generation of paraphrases from ambiguous logical forms. In *Coling'96.*

A. Stolcke. 1997. Linguistic knowledge and empirical methods in speech recognition. *AI Magazine*, 18(4):25-31.