A Statistical Approach to Language Modelling for the ATIS Problem

by

Joshua D. Koppelman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

Author ....
Department of Electrical Engineering and Computer Science
January 19, 1995

Certified by ....
Stephanie Seneff
Principal Research Scientist
Thesis Supervisor

Certified by ....
Salim Roukos
Manager of Language Modelling, IBM
Thesis Supervisor

Accepted by ....
F. R. Morgenthaler
Chair, Department Committee on Graduate Students

A Statistical Approach to Language Modelling for the ATIS Problem

by

Joshua Koppelman

Submitted to the Department of Electrical Engineering and Computer Science
on January 21, 1995 in partial fulfillment of the requirements for the
Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

The Air Travel Information Service (ATIS) is the designated common task of the ARPA Spoken Language Systems Program. The specified task is to build and evaluate a system capable of handling continuous and spontaneous speech recognition as well as natural language understanding in the ATIS domain. The goal of this research is to develop an effective natural language component for the complete system, to answer queries posed through text input instead of speech. We limit our scope to deal only with those sentences which can be understood unambiguously out of context (the so-called "Class A" queries). Specifically, we wish to use the training data to assign a probability distribution to the reference interpretation, the *NLParse*, which will minimize the observed perplexity of our test data. The decoder component of the finished system will use the natural language probabilities to select the most probable NLParse translations for a given English input. The NLParse translation can then be unambiguously converted to SQL to find the correct answer.

The first model we look at is a deleted interpolation trigram model on the NLParse, which ties low count parameters together to deal with sparse trigram data. Under the best parameter bucketing scheme, we observed a 15.9 bit per item test perplexity, with an average item length of 13.2 words. Since the NLParse is highly structured, it seems probable that models which use the information inherent in its structure will do better than those built only on the surface form. With this in mind, we have created a semantic tree structure which attempts to capture relationships between words and phrases in the NLParse. These trees constitute the training data for a series of improved language models, in which the probability of adding a node to a tree depends on the count of nodes with similar features in the training data. These models use a *maximum entropy* approach to maximize the entropy while satisfying the constraints imposed by the frequency of features in the training data. The best model uses this approach, and exhibits an average test perplexity of 14.1 bits per item. This represents a significant 1.8 bit reduction in perplexity over the trigram model benchmark.

Thesis Supervisors:

Salim Roukos, Manager of Language Modelling, IBM

Stephanie Seneff, MIT Laboratory for Computer Science Principal Research Scientist

Table of Contents

Acknowledgments

First and foremost, I would like to thank Salim Roukos and Stephen Della Pietra at IBM's Thomas J. Watson Research Center for their extensive help and guidance. The work presented here was conducted while I was an intern at T. J. Watson during the spring and summer of 1994, in conjunction with MIT's Computer Science internship program. Salim was my supervisor and mentor throughout my stay at IBM, and both he and Stephen were always available to answer the innumerable questions I had while learning about natural language and maximum entropy models. I would also like to thank Peter Brown and Robert Mercer, presently at Renaissance Technology, and Todd Ward, Mark Epstein, and Vincent Della Pietra at T. J. Watson for all of their help in resolving implementation issues. Finally, I wish to thank Stephanie Seneff, my thesis supervisor, who took time out of her busy schedule to offer her valuable input.

Joshua Koppelman

MIT, Cambridge, MA

January, 1995

1 Introduction and Background

This thesis concerns the development of a critical component of a system currently under development at IBM which, when complete, will be able to reply to spoken English queries with appropriate responses from a relational database. We have chosen to use the Air Travel Information Service (ATIS) as our domain, in large part because there has been a great deal of prior research in this area; it has been selected as the designated common task for evaluation of progress within the ARPA Spoken Language Systems (SLS) community [1]. The specified task is to build and evaluate a system capable of handling continuous and spontaneous speech recognition as well as natural language understanding in the ATIS domain, based on selected tables derived from the on-line Official Airline Guide (OAG, 1990). Users may ask questions about almost any issue which might arise in planning an airline voyage, including: lists of flights departing from or arriving at specified destinations, the fares for these flights, which meals will be served, the ground transportation provided between an airport and a city, and numerous others. This thesis specifically focuses on the ATIS2 task, which includes airports and airlines associated with 10 different US cities. Recently, the ATIS3 task has been defined to include more than 50 cities, with a corresponding increase in the number of airlines and airports.

As a consequence of the amount of research that has been done in the ATIS domain, there exists a large corpus of transcribed utterances spoken by users as they have attempted to solve particular scenarios within the domain. A large subset of this corpus has also been provided with annotated responses, including, most critical for our work, an *NLParse* reference interpretation, which is essentially a paraphrase of the original query in a more standardized form. A further advantage afforded by the prior research effort is the existence of a benchmark performance with which to compare the performance of our system, once it is fully functional.

The approach being adopted at IBM follows closely the statistical approach IBM has successfully used in the past in several other applications, most notably, language translation. In the ATIS context, we view the problem of "understanding" an English query as the need to translate from English to NLParse. A system which can automatically produce this translation will then be able to generate the SQL necessary to get an accurate response from the OAG table.

Our goal is to see whether standard statistical language modelling techniques can be applied successfully to the ATIS understanding task. We feel that there are many potential advantages to statistical language modelling. Specifically, we are able to minimize the domain specific aspects of the task, providing extensibility to broader problems; the entire process can be automated, assuming the availability of specifically labeled training material (e.g. the NLParse); and we avoid the need for human-generated templates and hand-written heuristic rules, both of which are time consuming and prone to error. However, it should be noted that we have not yet addressed how to incorporate discourse context into our framework. As a consequence, for our initial system we will restrict ourselves to those sentences which can be understood unambiguously out of context (the so-called "Class A" ATIS2 queries [2]).

In this chapter, we will first briefly describe other approaches that have been used for the ATIS domain. We will then outline in more detail the framework for the IBM approach to the ATIS task, stating clearly where the research of this thesis fits in. Finally, we will provide a brief review of other application areas in which statistical approaches have been successful.

1.1 Prior ATIS Efforts

The ATIS task touches on a wide range of issues, both in natural language and speech recognition, and it is being used as a standard for comparison and evaluation of the progress of the various groups involved. Interested groups meet yearly for the ARPA Speech and Natural Language Workshop, where progress is evaluated [3]. These groups have considered numerous approaches, bringing to bear speech and natural language techniques which have proven valuable in other applications. As a result, they have been able to demonstrate a query accuracy rate as high as ninety percent. An overview of a few of the various approaches follows.

CMU's spoken language understanding system, Phoenix, and its recent improvements are described in [4, 5]. The Phoenix system has separate recognition and parsing stages. The recognizer passes a single word string hypothesis to a frame-based parsing module, which uses semantic grammars to produce a semantic representation of the input utterance. The system attempts to divide the utterance into phrases which match word patterns associated with different slots in a frame. Frames are associated with various types of actions that can be taken by the system. The system is implemented as a top-down Recursive Transition Network chart parser for slots.

In MIT's ATIS system [6], a segment-based speech recognition component (SUMMIT) [7] produces a list of the top *N* sentence hypotheses, which are then filtered by a probabilistic natural language component (TINA) [8]. The recognition component uses a class n-gram language model to provide language constraints. TINA employs a parsing strategy which attempts to piece together parsable fragments. As in CMU's system, it uses a semantic frame representation to encode meaning, although in MIT's frame, syntactic structure is also encoded. The first step is to produce a parse tree for the input word stream. A second-pass treewalk through the parse tree yields a semantic frame, which is then passed to the back-end for interpretation.

AT&T's speech understanding system, Chronus [9], is based on the stochastic modelling of a sentence as a sequence of elemental units that represent its meaning. The representation of the meaning of a sentence consists of a relational graph whose nodes belong to some category of concepts and whose arcs represent relations between concepts or linguistic cases. A speech recognition component passes the single best word string to a *conceptual decoder*, which attempts to provide a conceptual segmentation. A simple pattern matching procedure then uses the conceptual segmentation to produce a corresponding set of values aligned to the concepts. Finally, the back-end translates the meaning representation into a database query.

BBN's HARC [10] spoken language system consists of a speech recognition component, BYBLOS, which interacts with their natural language processing component, DELPHI, through an N-best interface. DELPHI uses a definite clause grammar formalism, and a parsing algorithm which uses a statistically trained agenda to produce a single best parse for an input utterance. SRI's Gemini system [11] takes a linguistic approach to natural language understanding, in which a constituent parser applies syntactic, semantic, and lexical rules to populate a chart with edges containing syntactic, semantic, and logical form information.

1.2 The IBM Approach

The effort at IBM has only recently begun, and the task has initially been divided into five parts:

- A speech recognizer, which converts spoken queries to text queries.

- A translation model, which assigns a probability to a proposed alignment between an English query and an intermediate language query.

- A language model, which assigns a probability to a future event given a history in the intermediate language.

- A decoder, which searches for the sentence in the intermediate language which the language model and translation model together assign the highest probability.

- A back end, which deterministically converts from the intermediate language to SQL, to retrieve data from the OAG tables.

Here we focus on the natural language component, answering queries posed through text input instead of speech, and limited in scope to deal only with those sentences which can be understood unambiguously out of context. Ultimately, we wish to assign a probability distribution to the intermediate language (L), given an English query (E). This probability, $Pr(L|E)$, can be rewritten using Bayes' rule as

$$Pr(L|E) = \frac{Pr(E|L)\,Pr(L)}{Pr(E)} \qquad \text{(Eq 1.1)}$$

where $Pr(L)$ is the *a priori* probability of the intermediate language sentence. Typically, we wish to find the intermediate language sentence which maximizes $Pr(L|E)$. Since $Pr(E)$ does not depend on L, it is sufficient to maximize $Pr(E|L)\,Pr(L)$. The task of the translation model is to determine the probability distribution $Pr(E|L)$, while it is the language model's role to provide the distribution $Pr(L)$.

1.3 Statistical Modelling Research

The statistical approach to language modelling has been used with varying success in the past. IBM's machine translation system, Candide [12], which provides automatic translation from French text to English text, is designed in very much the same way as the spoken language system for ATIS which will contain the language model presented in this work. Its principle components are: a language model, which produces a probability distribution on the source language, English; a translation model, which produces the probability of a French sentence given the English (i.e. the probability of a French–English alignment); and a decoder, which searches for the French sentence that maximizes the product of the probabilities given by the translation model and the language model. Both the language model and the translation model are built exclusively from statistics taken directly from the training data. Candide's language model uses n-gram statistics (which will be explained in more depth shortly) in the same way in which we will use them. While each of these components is similar to the corresponding component in the ATIS system, each must overcome difficulties specific to the translation task.

Another application in which a statistical approach has been used successfully is in decision tree parsing. In [13], Jelinek *et al.* discuss the use of statistical models in *treebank recognition*, the task of automatically generating annotations for linguistic content on new sentences. Jelinek points out that there is no reason that a grammar needs to be used to construct the probabilistic model necessary for parsing. Instead, a method for constructing a statistical model for the conditional distribution of trees given a sentence *without a grammar* is presented.

In a restricted subset of the ATIS domain, researchers at AT&T have shown how statistical modelling can be used to train a translation model between a specified English grammar and an intermediate semantic language [14]. Using Equation 1.1, they have separated the task of learning associations between grammars into the task of building input and output language models, and of training a transducer for translation between the two. Building the necessary language models was not addressed in much depth, and we feel that the language models which we will present in this work could be used effectively in conjunction with their techniques.

In this first chapter, we defined the ATIS understanding task, and discussed a number of approaches which are currently being implemented. We discussed IBM's approach, and showed how it differs from those which are currently in use. We presented a few of its drawbacks, as well as some of its potential advantages. Finally, we motivated our statistical approach to the ATIS task by enumerating a few similar applications in which statistical methods have proven successful. In the next chapter, we will give a general overview of how different statistical language models are built and trained. In Chapter 3, we will take a closer look at the data the models will be built from, and the steps necessary to prepare the data for use. Chapters 4 and 5 will present a number of different language models as applied specifically to the ATIS task, and report on their effectiveness. Chapter 6 will discuss a way to enhance our language models using a known peculiarity in the data, and finally, Chapter 7 will summarize our findings, and suggest possible extensions to our work.

2 Statistical Language Modelling

In this chapter, we will give a general overview of how different statistical language models are built and trained. We will formulate a simple trigram model, built from frequency statistics on the training data, and we will discuss ways to deal with the ever-present difficulties with sparse data. Since the entire system is not yet complete, we are unable to see how the various language models work in tandem with the other finished components of the system, so we will define *perplexity* as our criterion for success. We will then present an effective method for estimating the parameters of the trigram model, realizing that some statistics are more reliable than others. Finally, we will discuss the maximum entropy framework for building language models, and present an algorithm for training such models.

2.1 Trigram Language Models

The simplest language model assigns a uniform probability across all the words in the source language (assuming a known vocabulary size), with the probability of the entire sentence equal to the product of the probability of the words. Under this model, each sentence is assigned a probability which is exponential in its length. A slightly more intelligent language model might assign each word a probability proportional to its observed frequency in a set of *training data* which is known to have similar characteristics to the data on which the model will be used. In general, a statistical language model uses training data to estimate the true *a priori* probability of words in the source language, given any information already known about their context.

The probability of a sentence $L = w_1 w_2 \ldots w_n$ is given by

$$Pr(L) = \prod_{i=1}^{n+1} Pr\left(w_i \middle| w_0 w_1 \ldots w_{i-1}\right) \qquad \text{(Eq 2.1)}$$

where w_0 and w_{n+1} are special boundary word tokens. A language model can take as much or as little information about the previously seen context (called the *history*) into account as it wants. Theoretically, the larger the history considered, the more accurate the model. In practice, little or no advantage has yet been gained by looking outside of a very local area. The success of the *trigram language model* [15] is a testament to this fact. The trigram language model predicts the probability of a word only considering the last two words in the history. A simple trigram model would assign each trigram a probability proportional to its frequency count in the training data:

$$Pr\left(w_i \middle| w_{i-2} w_{i-1}\right) = \frac{c\left(w_{i-2} w_{i-1} w_i\right)}{c\left(w_{i-2} w_{i-1}\right)} \qquad \text{(Eq 2.2)}$$

Given a large enough training set, this would be a very good approximation. In practice, there is rarely enough training data to ensure that each trigram is seen a significant number of times. It is a particularly bad idea to assign a probability of zero to *any* trigram. This problem of how to deal with sparse data proves to be a difficult one when building language models.

A more sophisticated trigram model offers the solution of backing away from low count trigrams by augmenting the estimate using bigram counts. The possible existence of rare bigrams suggests backing off even further, to the unigram and uniform distributions. The deleted interpolation trigram model assigns a probability to each trigram which is the linear interpolation of the simple trigram, bigram, unigram, and uniform models:

$$Pr(w_i|w_{i-2}w_{i-1}) = \lambda_3 \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})} + \lambda_2 \frac{c(w_{i-1}w_i)}{c(w_{i-1})} + \lambda_1 \frac{c(w_i)}{N} + \lambda_0 \frac{1}{V}$$

Eq 2.3

Here N is the number of words in the training corpus, V is the size of the vocabulary, and the lambdas, the relative weights given to each of the models, must sum to 1. A positive λ_0 ensures that no trigram will be assigned a zero probability.

2.2 Perplexity as Performance Measure

It is useful to have a measure of the effectiveness of a language model which is independent of the components of the system which will ultimately use it. A frequently used measure of a language model's performance is its *perplexity* [15] with respect to a test corpus (T). If $\tilde{P}(T)$ is the frequency distribution observed in the test corpus, and $P(T)$ is the probability distribution predicted by the language model, then the perplexity (PP) of the language model with respect to T can be written in terms of the cross entropy (H) of $\tilde{P}(T)$ and $P(T)$:

$$H\left(\tilde{P}(T),\ P(T)\right) = -\sum_{x \in T} \tilde{P}(x) \log P(x) \qquad \text{(Eq 2.4)}$$

$$PP(T) = 2^{H\left(\tilde{P}(T),\ P(T)\right)} \qquad \text{(Eq 2.5)}$$

If a language model assigns a zero probability to *any* event which is seen in the test corpus, the perpexity is infinite. It should be noted that a lower perplexity does not necessarily imply a lower error rate in the resulting system, though this is often the case. In training many of the models presented in this paper, we assume a correlation, and implement algorithms which attempt to minimize the perplexity with respect to the training data. In fact, we use perplexity as our sole criterion for success.

2.3 Parameter Estimation and Bucketing

With a large enough training corpus, λ_3 in Equation 2.3 can be set very close to 1. For sparse data, we need a more precise way to determine the best values for the lambdas which are used to smooth the models. We use the Estimation Maximization (EM) algorithm proposed by Baum [16] to train the values of the lambdas to minimize the perplexity of the deleted interpolation language model on the training data. It assumes the existence of a set of *heldout* data (H) independent of the training corpus used by the individual models being smoothed together. The EM algorithm is an iterative algorithm which is guaranteed to decrease the model's perplexity on every iteration. To calculate the weight λ_i to be assigned to the probability distribution P_i:

1. Set the iteration variable, $j = 0$
2. Assign initial guesses to λ_i^0, ensuring $\sum_i \lambda_i^0 = 1$
3. For each model i to be smoothed, set $c_i = \sum_{e \in H} \frac{\lambda_i^j P_i(e)}{\sum_k \lambda_k^j P_i(e)}$
4. Set $\lambda_i^{j+1} = \frac{c_i}{\sum_k c_k}$
5. Set $j = j + 1$
6. Repeat from 3.

The algorithm might typically be iterated until the incremental decrease in perplexity drops below a given threshold.

The trigram model in Equation 2.3 is still very simple; since the training count is an estimator of the reliability of the statistic, those trigrams with large training counts should have their own lambda, while small count trigrams can be grouped together. If we gave *all* trigrams their own lambdas, the model would overtrain; it would do very well on the heldout data, but not as well on other data. We would like to leave some flexibility. The idea is that we put trigrams with similar reliability (i.e. the same training count) in the same *bucket*, and assign a vector of lambdas to each bucket, to be trained independently with the EM algorithm. By defining a bucketing function b, parameterized over the bigram and unigram counts, the model becomes:

$$Pr\left(w_3 \middle| w_1 w_2\right) = \sum_i \lambda_i(b(c(w_2), c(w_1 w_2)))\ P_i\left(w_3 \middle| w_1 w_2\right) \qquad \text{(Eq 2.6)}$$

Typically, trigrams are added to a bucket until the number of training events which are assigned to that bucket is greater than some constant number, and then a new bucket is started. We call this bucketing scheme "wall of bricks" bucketing:

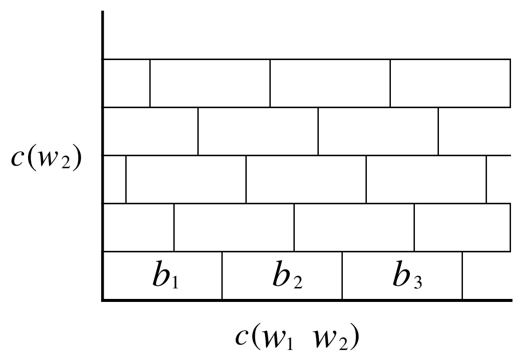


Figure 1. "Wall of Bricks" Bucketing

2.4 Maximum Entropy Models

A detailed treatment of the maximum entropy approach to natural language processing can be found in Berger *et al* [17]. A maximum entropy model begins with an initial distribution which represents everything that is known *a priori* about the source being modelled, and a set of *constraints* representing the information to be modelled. The concept of maximum entropy is to find the distribution which has the smallest Kullback-

Leibler distance from the initial distribution, and which satisfies the constraint set while assuming no other knowledge about the source. If Q is the initial distribution, P is a distribution which satisfies the constraint set, and X is the space of futures to be predicted, the Kullback-Leibler distance between P and Q is given by:

$$D(P\|Q) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \qquad \text{(Eq 2.7)}$$

2.4.1 Feature Functions and Constraints

A feature function $f(x, y)$ is a binary function over the set of histories x and futures y. A feature function which has the value 1 for some history $x = w_1\ w_2\ ...\ w_m$ and some future $y = w_{m+1}$ is said to *fire* on event $w_1\ w_2\ ...\ w_{m+1}$. If we observe that a feature f_i fires n times on the training data, we wish to impose the constraint that the language model will predict that f_i fires n times on the same data. The maximum entropy framework allows us to do this effectively. If T is the training corpus, and Pr is the maximum entropy probability distribution, then for each feature f_i in the feature set, we add the constraint

$$M_i = \sum_{x,y} Pr(x,y) f_i(x,y) = \frac{1}{|T|} \sum_{(x,y) \in T} f_i(x,y) = d_i \qquad \text{(Eq 2.8)}$$

Here we are equating the *model expectation*, M, with the *desired expectation*, d. Given an initial distribution Q, we can now define the maximum entropy distribution as follows:

$$Pr(y|x) = \frac{Q(y|x)}{Z(x)} e^{\sum_i \lambda_i f_i(x,y)} \qquad \text{(Eq 2.9)}$$

Where Z is defined so that $\sum_y Pr(y|x) = 1$. In order to normalize in this way, it is necessary to know the set of all possible futures for any given history. For the applications of maximum entropy in this paper, the set of possible futures is always known; the intermediate language is well defined. For other applications, modelling speech for example, it is very difficult to sum over all possible futures, so the model must be modified. A complete treatment of the necessary modification is given in Lau [18].

All that is left is the decision of which feature functions to include, and an explanation of how to train the model parameters λ_i. The set of feature functions is equivalent to the power set of legal histories and futures, and hence, the space of feature functions is phenomenally large. It is not feasible to include all possible feature functions in our feature set. In addition, if we add too many features, we run the risk of overtraining the model on the training data, resulting in an inability to generalize to new data. One of the most difficult tasks in designing a maximum entropy language model is deciding which features to add. Berger *et al* [17] discusses a greedy algorithm for automatic feature selection, in which an attempt is made to incrementally add the feature which will decrease the model's perplexity most.

2.4.2 Improved Iterative Scaling

How do we calculate the model parameters λ_i in Equation 2.9 to minimize the Kullback-Leibler distance from the initial distribution? While it is not possible in general to find a closed form solution, Darroch and Ratcliff [19] have proposed a method of generalized iterative scaling to determine the solution numerically. More recently, Della Pietra [20] has developed an improved iterative scaling algorithm which allows feature functions whose sum is any non-negative real-valued function. This allowance has significant practical performance implications.

The improved iterative scaling algorithm presented below uses the following two function definitions (assuming n is the number of features in the feature set):

$$k(x,y) = \sum_{i=1}^{n} f_i(x,y) \qquad \text{(Eq 2.10)}$$

$$p_\beta(y|x) = \frac{Q(y|x)}{Z_\beta(x)} \prod_{i=1}^{n} \beta_i^{f_i(x,y)} \qquad \text{(Eq 2.11)}$$

As before, Z is defined as the normalization factor which allows p_β to sum to 1 over all possible futures y. Note that Equation 2.11 is just another way to write Equation 2.9. The improved iterative scaling algorithm can now be summarized as follows:

1. Initialize β_i for $1 \le i \le n$

2. Calculate the desired expectation from the training corpus T for each feature:

   $$d_i = \frac{1}{|T|} \sum_{(x,y)\in T} f_i(x,y)$$

3. For $1 \le i \le n$, let δ_i be the solution to the following equation:

   $$\sum_{x,y} p_\beta(y|x)\, \delta_i^{k(x,y)} f_i(x,y) = d_i$$

4. For $1 \le i \le n$, update $\beta_i = \delta_\iota \cdot \beta_i$

5. Repeat from 3 until all β_i converge, or until the change in perplexity is within a preset threshold.

6. For $1 \le i \le n$, $\lambda_i = \ln \beta_i$

The solution to the polynomial in step 3 can be found numerically by using the well-known Newton-Raphson method for root-finding. This algorithm is guaranteed to converge to the correct solution (i.e. it determines the values of λ_i which make Equation 2.9 the maximum entropy solution), assuming that a solution which satisfies all of the constraints exists. Unfortunately, there is no provable bound on the number of iterations necessary for the algorithm to converge, so a decrease in training perplexity is often used as the stopping criteria.

3 Data Preparation

Data collection and distribution for the ATIS task is described in [19, 20, 21]. The data have been obtained through the National Institute of Standards and Technology (NIST), which makes it available to interested sites through anonymous FTP. The data consist of a corpus of English queries, together with a reference interpretation (called the *NLParse*), a reference SQL, and a reference answer. The NLParse commands reflect conventions on how to interpret an ATIS query, in a canonical form specified by an established context-free grammar. A sample English query and its corresponding NLParse interpretation are given in Figure 2. Software to convert from the NLParse to SQL capable of obtaining the desired information from the OAG database has been completed and is in use at IBM.

English:	List all flights from Baltimore to Atlanta after noon Thursday nonstop.
NLParse:	List flights from Baltimore and to Atlanta and flying on Thursdays and leaving after 1200 and nonstop.

Figure 2. Sample Query with NLParse Interpretation

A primary test corpus of 1800 items has been separated from the 5600 Class A items obtained from NIST. This test corpus is being held apart until the entire system is ready to be tested as a whole. It is from the remaining 3800 sentences that all of the training and testing of language models were done. These 3800 items were further separated into three categories: *training* (3000 items), *heldout* (400 items), and *test* (400 items).

3.1 Preprocessing

The NLParse interpretation set supplied by NIST was not originally meant for distribution, and a large amount of human and automatic cleanup needed to be done to remove ambiguity and ensure consistency between items. To aid in language modelling, we have written a tagger, which replaces numbers, cities, airlines, airports, codes, days, and dates in the English and NLParse files with the tags NUM, CITY, AIR, ARP, CODE, DAY, and DATE, respectively. Using these categorizations allows statistics to be gathered about groups of tokens which are used in the same syntactic positions, but which individually are seen infrequently in the data. For example, the date "7/12/91" may be seen only once in the training data, but after tagging, the tag "DATE" may appear hundreds of times. Items with multiple tags of the same type have identifiers appended to permit easy untagging. Figure 3 shows an example English and NLParse item before and after tagging.

English: What time does Continental depart from Boston to San Francisco?

Tagged English: what time does AIR_1 depart from CITY_1 to CITY_2

NLParse: List departure time of Continental flights from Boston and to San Francisco

Tagged NLParse: list departure_time of AIR_1 flights from CITY_1 and to CITY_2

Figure 3. Tagged Items

A language model can be more confident of its statistics about the tags than about the original items. In the final system, a component will be added to the back end which replaces tags with canonical spellings of the corresponding words from the English. At present, a finite state machine searches for all conceivable ways of writing instances of each tag type, but it is not infallible, and it will be replaced by a probabilistic tagger in the near future.

3.2 Semantic Trees for NLParse

At first, we thought that it might be possible to statistically generate an SQL query from the English input. This idea was quickly discarded for numerous reasons (e.g., the average length of an SQL query is much greater than the English which corresponds to it, making it difficult to build an effective translation model). Thus the first pass language models will be built using the NLParse as an intermediate language.

Since the NLParse is highly structured (it is defined by a set, if complicated, context-free grammar), it seems probable that models which use the information inherent in its structure will do better than those built only on the surface form. With this in mind, we have created an information tree structure which attempts to capture relationships between words and phrases in the NLParse. Rather than force the language models to learn the complicated grammar which specifies the NLParse, the trees are structured to conform to the more general grammar given in Figure 4. The tree structure is defined by the grammar; every production is equivalent to a parent-child link in the tree. The label of each node is the unique string on the RHS of the production. The children of each node are each of the non-terminals on the RHS, in the order specified. To resolve any ambiguity in ordering of children (for example, in the ordering of modifiers), the node with the higher parent-child bigram frequency is placed first. Finally, homonyms in the NLParse have been relabeled in the node labels. For a complete list of the label vocabularies used, see Appendix A.

Note that since the non-terminal *Table* appears on the RHS of the *Modifier* production, the NLParse, and hence the trees, can be arbitrarily complex. Also, the grammar given allows an arbitrary number of tables to be accessed using "along-with," even though the current specification of the ATIS problem allows cross-referencing of only two tables at the same time. This is provided for extensibility to future ATIS versions.

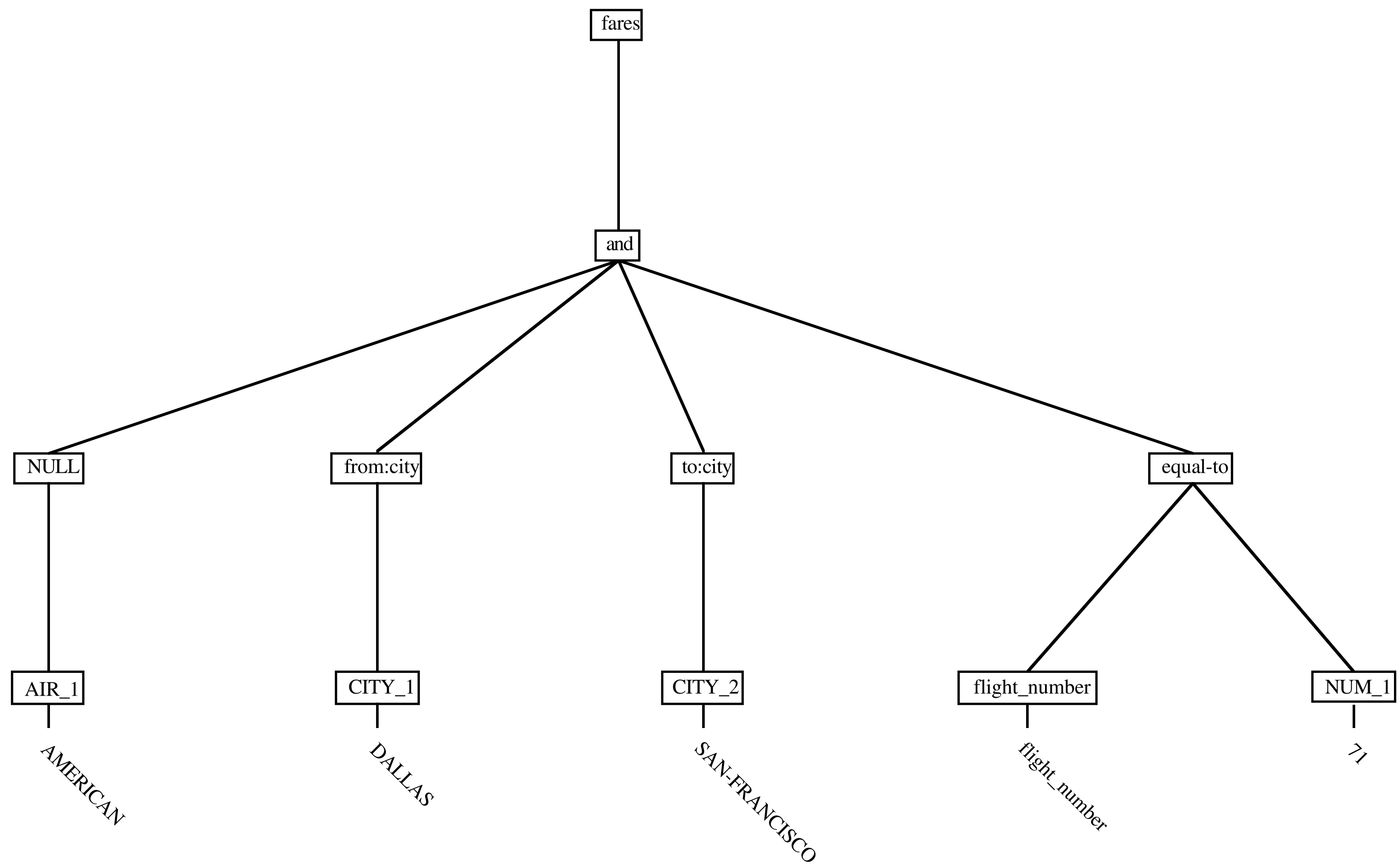
Figures 5 and 6 show a simple and more complicated example tree. We have written code to automatically transform a surface NLParse item into a semantic tree item, and back again. Ultimately, both the language and translation models will be built from the information trees, and the decoder's output trees will be deterministically converted into NLParse, and then to SQL.

```
<Along-With>      ::  ( "Along-With" <Column-Table> + ) ?  <Column-Table>

<Column-Table>    ::  <Grouped-By>  |  "Extract"  <Grouped-By>  <Features>

<Features>        ::  "Features"  <Feat-Op> +

<Feat-Op>         ::  feature_op  <Terminal> +

<Grouped-By>      ::  <Table>  |  "Grouped-By"  <Table>  <Terminal>

<Table>           ::  table_name  ( <And-Phrase> | <Or-Phrase> ) ?

<And-Phrase>      ::  "and"   (  <Or-Phrase> | <Modifier> ) +

<Or-Phrase>       ::  "or"   ( <And-Phrase> | <Modifier> ) +

<Modifier>        ::  terminal_op  <Not> ?  <Terminal> +                    |
                      terminal_op  <Not> ?  <Table>                         |
                      compare_op  <Not> ?  <Terminal>  <Terminal> +         |
                      compare_op  <Not> ?  <Terminal>  <Feat-Op>  <Table>

<Not>             ::  "Not"

<Terminal>        ::  terminal_name
```

Where each of the following denotes a string from a set vocabulary:

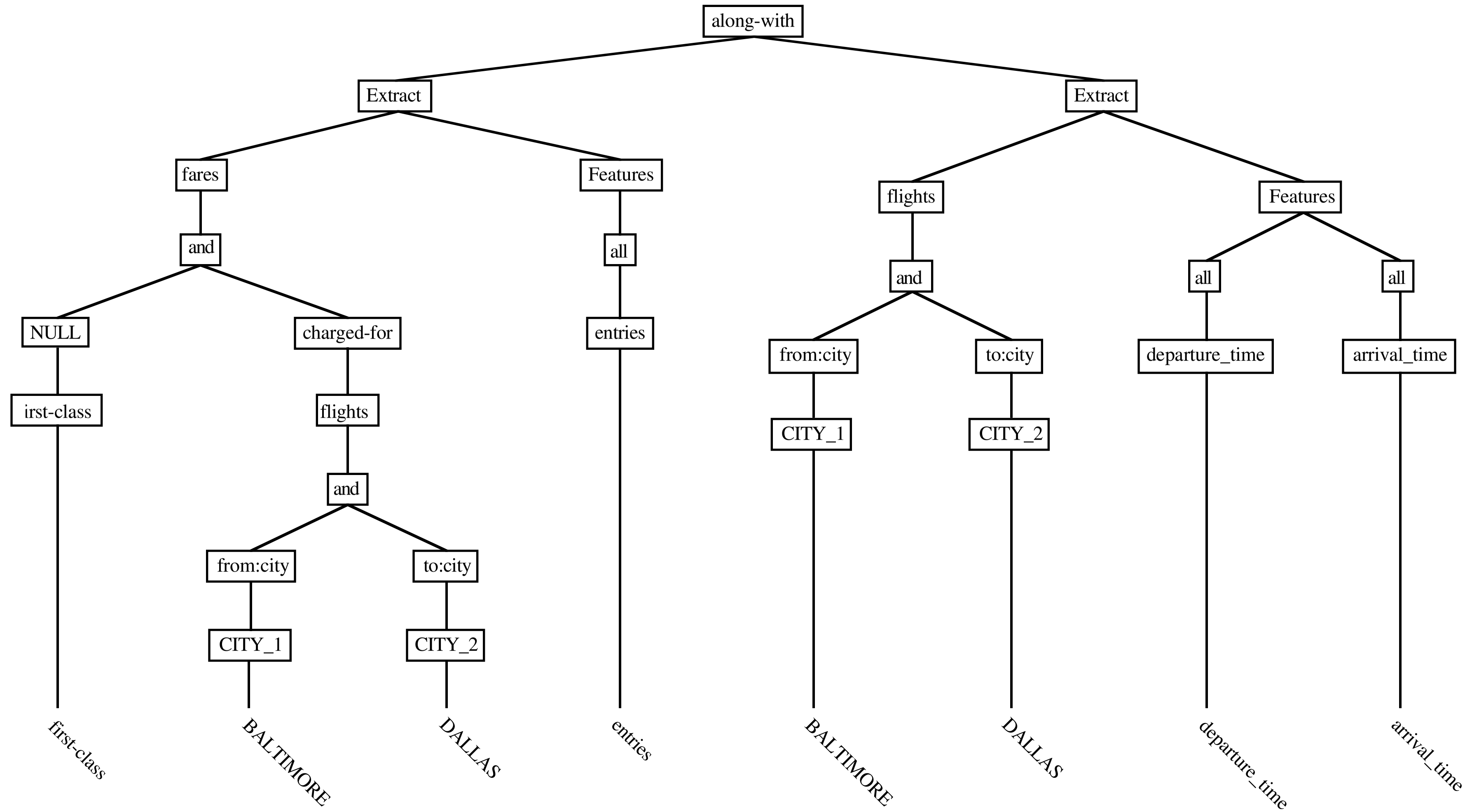
table_name	from	flights, fares, ...
compare_op	from	less-than, equal-to, between, ...
feature_op	from	the-maximum, the-minimum, the-number-of, any, all
terminal_op	from	cheapest, from, arriving, ...
terminal_name	from	CITY_1, DATE_1, arrival_time, departure_time, ...

Figure 4. Complete NLParse Semantic Tree Grammar



English:	How much does the American Airlines flight seven one from Dallas to San Francisco cost?
Tagged English:	how much does the AIR_1 flight NUM_1 from CITY_1 and to CITY_2 cost
NLParse:	List American fares from Dallas and to San Francisco and whose flight number is 71
Tagged NLParse:	list AIR_1 fares from CITY_1 and to CITY_2 and whose flight_number is NUM_1

Figure 5. Simple NLParse Semantic Tree



English:	Show me the prices and times for first class travel from Baltimore to Dallas
Tagged English:	show me the prices and times for first class travel from CITY_1 to CITY_2
NLParse	List first class fares charged for flights from Baltimore and to Dallas along with departure time and arrival time of flights from Baltimore and to Dallas
Tagged NLParse:	list first class fares (charged for flights from Baltimore and to Dallas) along with departure_time and arrival_time of flights from Baltimore and to Dallas

4 Trigram Language Models

Each of the language models we build will be trained on the training set, smoothed with the heldout set (if necessary), and tested on the test set. If smoothing isn't necessary, the heldout data will also be used to train. For each model, the average per item entropy for each data set will be reported; it is this that we use as a gauge to determine how effective the model is. The model with the lowest test entropy will eventually be used in the final system.

4.1 Surface Trigram Models

The first language model we've built is a deleted interpolation trigram model on the tagged surface NLParse, using the "wall of bricks" scheme for bucketing lambdas. The results for this model (and the other two models described in this section) are presented in Figure 7. Due to the relative success of the trigram model in other applications, we used this model as the benchmark for comparison with the other models we have built. In particular, we hoped to improve upon the test entropy of 15.9 bits per item. Since the average sentence length of the surface NLParse is 11.0 words, the decoder will have an average of 2.7 weighted options for each word.

The surface trigram model has some serious deficiencies for this particular application. First and foremost, it does not explicitly use any of the information which is know *a priori* about the source language, the NLParse. As a result, the model allows (i.e. assigns positive probabilities to) items which do not conform to the NLParse grammar, and hence are not valid NLParse items. A more intelligent model would realize this, and assign these items zero probability (allowing normalization over the smaller remaining space). To test just how deficient this model is, we have randomly generated NLParse sentences from the trigram probability distribution, and found that only 58% of the resulting sentences were in a valid NLParse format.

One of the reasons that this model is so poor at generating valid NLParse is that important words are frequently more than two words back in the history, and thus outside of the trigram model's scope. One example of this problem is parentheses, which are used to remove ambiguity in the cleaned version of the NLParse. The trigram model has extreme difficulty predicting a close parenthesis, since the matching open parenthesis is almost always more than two words earlier. Another word which is often more than two words back in the history is the *table* which needs to be referenced in the database. The table is the most important part of any query, and each table has its own set of modifiers and features associated with it. It doesn't make sense, for example, to ask about the "meal_description" from the "flights" table, yet the trigram model gives a non-zero probability to "meal_description" even when "flights" is nowhere in the sentence. In later models, we are sure to allow the models to ask questions about the table whose scope is currently active.

	Training	Heldout	Test
Items	3000	392	376
Surface NLParse	12.6800	13.4784	15.9474
Flat tree, with parens	18.9160	18.7158	20.3842
Flat tree, without parens	11.5749	12.6873	15.6387

Figure 7. Trigram model per item entropy

All subsequent language models will be built on the semantic tree form of the training and heldout data. The models generate a probability distribution on the space of semantic parse trees, which can be mapped onto the NLParse space (i.e. there is a surjective function from the semantic trees to the NLParse). The next language model we consider is a trigram model on the prefix traversals of the information trees, with parentheses added where necessary to ensure that the trees can be recreated from the flattened string. This model has as a benefit that more important information (the table, for example) is generally higher up in the tree, and therefore will appear before the less important details below it. As can be seen in Figure 7, this model fails miserably in comparison with the trigram model. It still suffers from the problem that trigram models cannot learn to balance parentheses, and a significant number of parentheses need to be added to disambiguate the traversals.

It would be nice to know the entropy of the previous trigram model without having to worry about the problems presented by the parentheses. The last surface language model we have built, therefore, was trained on the prefix traversal of the semantic trees *without* adding the parentheses needed to prevent ambiguity. The result is a lower perplexity than even the trigram language model, though only barely. Unfortunately there is no corresponding language model in the semantic tree space, so this model would leave it up to the decoder to decide which interpretation of the traversal is the most probable. Still, that such a simple use of the semantic trees is able to beat the trigram model is encouraging.

4.2 Tree-based Interpolation Models

The first tree-based language model we present is a generative deleted interpolation model on the semantic parse trees, where the probability of adding a node depends on the count of nodes with similar features in the training data. The nodes are generated in the same order as a prefix traversal of the tree. We considered different kinds of features, all of which involve only very local information. One of the problems with all the previous models is their inability to use statistics about important words which may appear further back in the history. Rather than using the previous two words from the prefix traversal, in this model a node's probability is related to the training frequency of its parent, and of the table in whose scope the node exists. Both of these pieces of information are always generated before the node being predicted, so their training statistics are available to the model. If n[i] denotes the ith child of n, and T is a function which returns the table under whose scope its argument exists, then the probability P_s of generating a subtree beneath node n is given by the recursive function:

$$P_s(n) = \prod_i Pr(n[i] | T(n), n)\, P_s(n[i]) \quad Pr(stop | T(n), n) \qquad \text{(Eq 4.1)}$$

Here $Pr(n|T, P)$ is the probability the model assigns to node n in the presence of table T and parent P. The product is taken over each of the children of n. Note that the model must predict a special *stop* child before continuing on to the next node. This is used as an alternative to first predicting the number of children each node has. The probability of a tree item is now simply $Pr(root \mid none, none) P_s(root)$.

Our first attempt at a trigram model using only the table and parent as context was to specialize Equation 2.3:

$$Pr(n | T, P) = \lambda_3 \frac{c(T\,P\,n)}{c(T\,P)} + \lambda_2 \frac{c(P\,n)}{c(P)} + \lambda_1 \frac{c(n)}{N} + \lambda_0 \frac{1}{V} \qquad \text{(Eq 4.2)}$$

The results of using this model, however, were abysmal, primarily because of its incompetent handling of the *stop* node. We need to allow the model to look at the number of siblings already predicted, to more effectively determine when to predict *stop*. Equation 4.1 was updated accordingly.

Another significant improvement was found by reconsidering the normalization factors of the component models. These factors can be made more accurate by incorporating information which we know *a priori* about the NLParse grammar, but which isn't reflected in the semantic tree grammar in Figure 4. Unfortunately, this involves encoding information specific to the current ATIS task, something which we had wished to avoid to allow easier application to future natural language understanding problems[1]. By implementing a function L, which maps a context to the set of futures permissible in that context, Equation 4.2 becomes:

$$Pr(n|T,P) = \begin{cases} Pr'(n|T,P,L(T,P)) & \text{for } n \in L(T,P) \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Eq 4.3)}$$

$$Pr'(n|T,P,\mathrm{L}) = \lambda_3 \frac{c(T\,P\,n)}{\sum_{n' \in \mathrm{L}} c(T\,P\,n')} + \lambda_2 \frac{c(P\,n)}{\sum_{n' \in \mathrm{L}} c(P\,n')} + \lambda_1 \frac{c(n)}{\sum_{n' \in \mathrm{L}} c(n')} + \lambda_0 \frac{1}{|\mathrm{L}|}$$

Eq 4.4

The results of training this model with only *one* bucket of lambdas is given in Figure 8 (along with the previous figures for convenience). These results are still very bad. Upon examination of the resulting distribution, we discovered that this model does not respect the canonical child ordering imposed upon the semantic trees. For example, the modifier "from:city" always precedes the modifier "to:city", when both are present in a tree (this stems from the fact that the bigram "and, from" is seen more frequently than "and, to" in the training data). But the model assigns a non-zero probability to the tree containing these modifiers in the reverse order. To fix this situation, we modify L, and smooth in another

[1] This is one of only two parts of the language model component which are specific to the ATIS 2 task. The other one is the tagger.

distribution, the 4-gram distribution using as context the parent, table, and *left-sibling* (S). The new model is given by Equations 4.5 and 4.6.

$$Pr(n|T,P,S) = \begin{cases} Pr''(n|T,P,S,L(T,P,S)) & \text{for } n \in L(T,P,S) \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Eq 4.5)}$$

$$Pr''(n|T,P,S,\mathrm{L}) = \lambda_4 \frac{c(T\,P\,S\,n)}{\sum_{n' \in \mathrm{L}} c(T\,P\,S\,n')} + Pr'(n|T,P,\mathrm{L}) \qquad \text{(Eq 4.6)}$$

As can be seen by the results in Figure 8, even using only one bucket for the lambdas, this model beats the NLParse trigram model on the test corpus by almost one bit, reducing the perplexity by half. The addition of the follow set L was instrumental to this improvement. It is this model which we will use as the initial distribution for the maximum entropy model presented in the next section[2].

	Training	Heldout	Test
Items	3000	392	376
Surface NLParse	12.6800	13.4784	15.9474
Flat tree, with parens	18.9160	18.7158	20.3842
Flat tree, without parens	11.5749	12.6873	15.6387
Tree Context Trigram	19.2082	20.1576	21.4704
Tree Context 4-gram	11.7567	13.5918	15.0037

Figure 8. Entropy of Deleted Interpolation Models

[2] It was thought that the maximum entropy approach would be much more effective for smoothing than any complicated bucketing scheme, so we did not implement the full "wall of bricks" bucketing for this model.

5 Maximum Entropy Models

In this chapter we will modify the maximum entropy framework presented in Section 2.4 to allow us to generate maximum entropy models on semantic trees. Throughout this chapter, Q will refer to the initial distribution needed as a starting point for the maximum entropy approach. Frequently, maximum entropy models are built using a uniform distribution as the initial distribution. We will instead use one of the tree-based interpolation models derived in the previous Chapter as our starting point.

This is especially useful because it allows us to incorporate knowledge of the NLParse grammar which does not appear in the more general semantic tree grammar in Figure 4. This knowledge is embodied in the L function, which allows us to assign zero probability to certain futures in certain contexts. By assigning a zero probability to certain events in the initial distribution, Equation 2.9 guarantees that those events will also have zero probability in the maximum entropy distribution. If we had instead used a uniform distribution as the initial distribution, the maximum entropy model would assign non-zero probabilities to events that could not possibly occur.

5.1 Event Dumping

We have already shown how a generative language model can be built for the semantic parse trees by starting at the root and predicting each node in a prefix ordering (Equation 4.1). The maximum entropy model will be designed in the same way; each node will be considered separately, appearing in the future position exactly once. A single feature function may fire multiple times on any one tree, but only once for each *event*. An event is a history-future pair, so a feature is a binary function on events. Looked at from this perspective, the training corpus is just a set of events, and the tree to which an event belongs has no real significance to the language model. The first step in building our maximum entropy model is dumping the events of each of the data sets. An example tree and the events it generates are given in Figure 9.

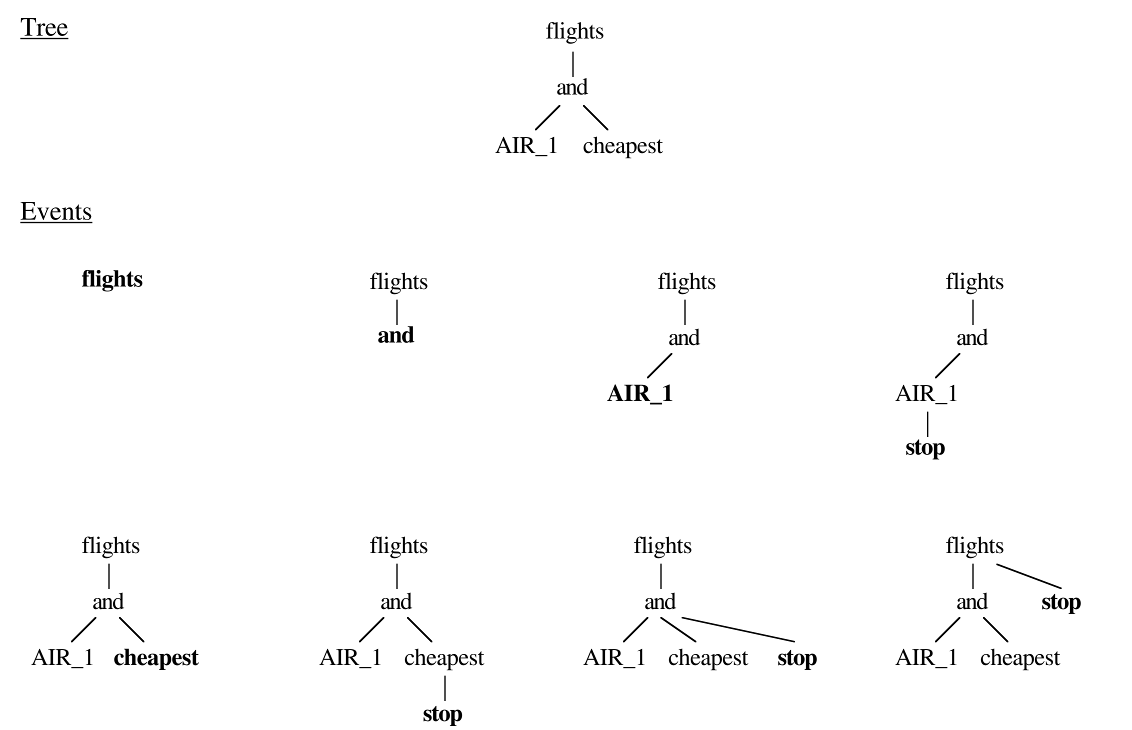


Figure 9. Sample event dump

The future is always the bottom rightmost node (in bold), and the history is the rest. *Stop* nodes are again treated differently, indicating that the given node has no more children. We must be sure to include features in our feature set which will allow the model to accurately predict the *stop* node.

5.2 Feature Selection

We now come to the most difficult part of building a maximum entropy model — selecting the salient features. We certainly want to include features which reinforce the n-gram statistics used to create our initial distribution. We also need to include features on the number of children different nodes have in different contexts, to accurately predict the *stop* node. Yet if we include only these features, we will not be using the most powerful aspect of the maximum entropy framework: its ability to incorporate many different kinds of information simultaneously. In particular, one nice feature of the maximum entropy training algorithm is that it does not require that features be disjoint; i.e., one feature may be a generalization of another feature.

Before selecting the features, we need to specify a language in which feature functions can be described. At first glance, it is not so clear how to implement a function whose domain is a set of trees along with some rules on where to look for "important" information (like the table scope). In the next section, we present a feature specification language built around the semantic parse tree formalism. Even given a specification language, however, it is not feasible to specify all of the features we will need by hand. For example, we will need a separate feature for *each* trigram which is seen a nontrivial number of times in the training data. Rather than listing what amounts to hundreds of features, we need a method to automatically collect relevant features. Section 5.2.2 describes a feature *pattern* language, which will allow us to specify higher level ideas about what kind of features to include[3].

5.2.1 Feature Specification Language

We need to create a specification language which is rich enough to allow us to describe features to test any pertinent characteristic of an event, but which will allow us to efficiently decide which events the feature will fire on. To this end, we decided that a feature must contain two items: a subtree describing information recently seen in the history (those nodes spatially local to the future node), and a flat context which would describe information that might be much further away. For our purposes, the context is just the table whose scope is currently active, but features could easily be specified which describe different non-local characteristics. A feature's *table* field may either contain a table name, in which case the feature will fire only for events with a future in that table's scope, or the special *any* value (which we denote with a question mark), which will cause the feature to disregard table information.

Subtree matching is slightly more complicated. Basically, a feature's subtree component will match if it can be overlayed onto an event, starting at the future node (the bottom rightmost one). Aligned nodes match if they have the same label, or if the feature's node has the *any* label. Subtree matching can be most easily understood through example. Figure 10 gives three sample features, and shows which events from Figure 9 cause the feature to fire. In the first example, the event matches even though it has a node labeled "cheapest". Our features currently can only be used to specify which nodes must appear,

---

[3] We use the feature pattern method as an alternative to the incremental feature selection method presented in Berger *et al* [9]. The latter is very computationally intensive, and it is not clear how to generalize it for use with tree based feature functions.

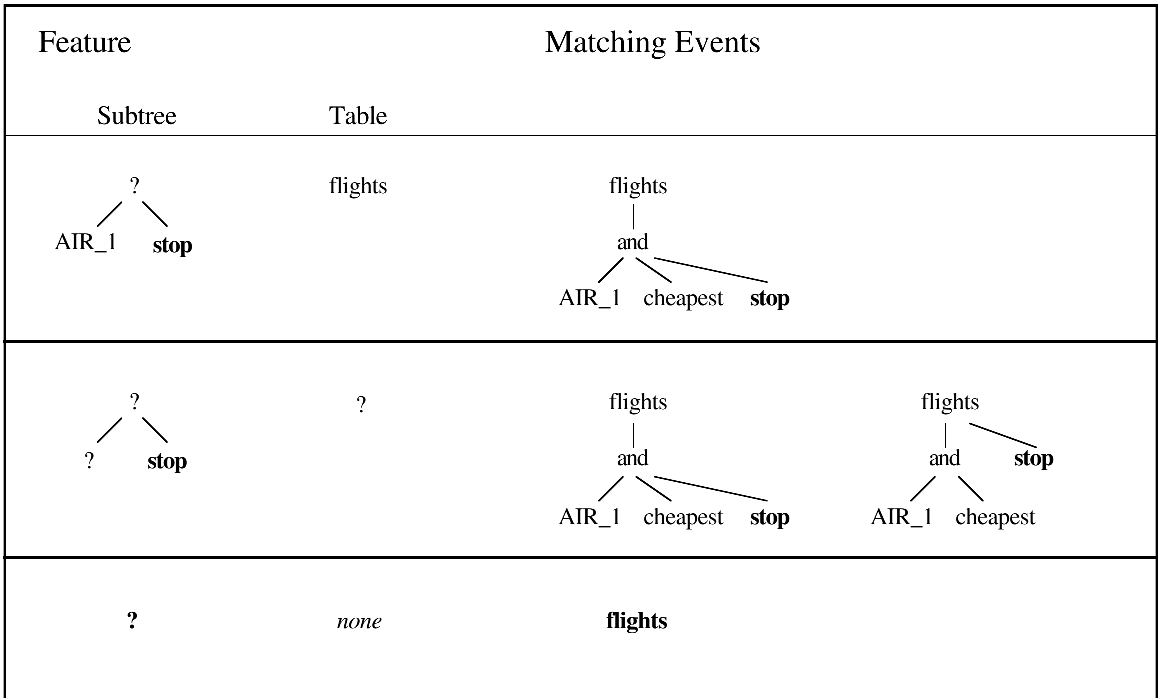


Figure 10. Sample features I

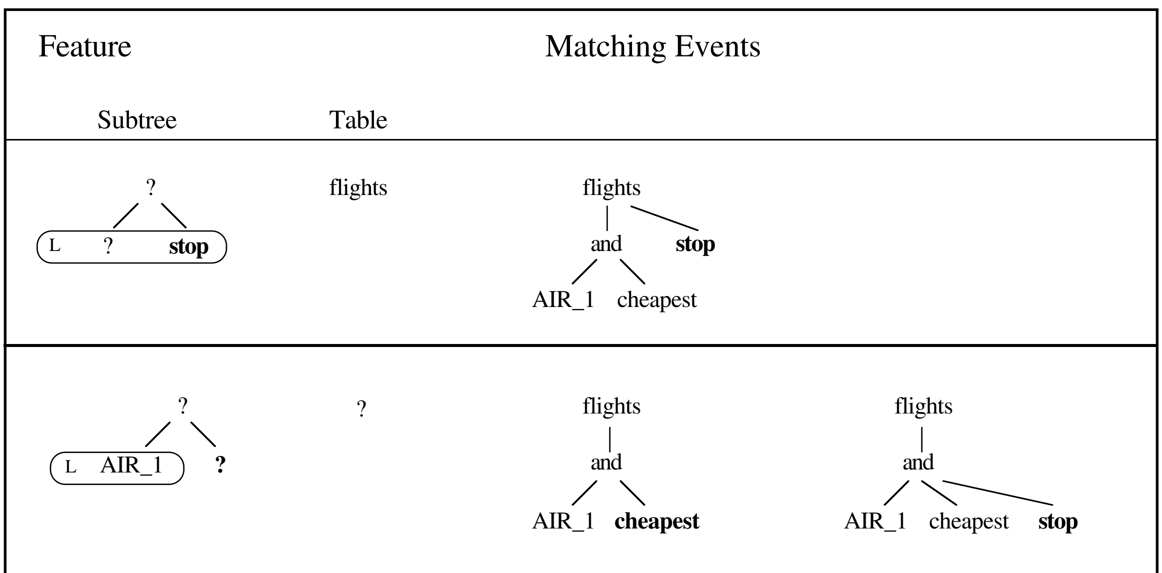


Figure 11. Sample features II

not which nodes cannot appear. In the second example, notice which events do *not* match. The *any* label will match any single node, but it will not match if no node exists. The third example shows that only the root node has *none* as it's table.

As described, the specification language allows us to effectively ask questions about tables and parents but not, for example, about siblings in specific positions. The interpolation models presented in Section 4.2 show that it would be useful to collect statistics about the sibling immediately to the left of the future node, but so far this isn't possible in our language. To allow this, we add the ability to "staple" nodes in the feature subtree to one side or the other. Stapling to the left side (denoted by the presence of 'L', the left anchor), forces nodes to appear *exactly* as specified on the left, with no additions. Since children are ordered by frequency, it may be useful to ask questions about the first few siblings. Similarly, nodes can be stapled to the right side (denoted by the presence of 'R', the right anchor), which is primarily useful in specifying immediate siblings of the future node. The future node is implicitly stapled to the right side. Figure 11 contains two examples of stapling. Notice that the feature in the first example will only fire for events where the parent of the future has exactly one child. We allow the use of the '#' character as alternate syntax for this idea (e.g. '#1' would denote one child). This syntax will become more important when we discuss feature patterns, below.

Our feature specification language has a number of shortcomings. One is its inability to specify negation, as noted earlier. Another is that there is no way to specify a feature which contains a disjunction of different characteristics. Extending the feature language is one candidate for possible improvement of our maximum entropy model. However, we feel that the language is versatile enough to allow the model to incorporate statistics about a sufficiently wide spectrum of event characteristics.

5.2.2 Feature Patterns

Going through all of the training data by hand to decide which features to include would be very tedious and imprecise. Instead, we would like to be able to specify the different kinds of features we want to have present, and then accumulate those features automatically. We would also like to have a mechanism which will allow us to filter out those feature which do not fire frequently on the training data; we don't want to include constraints in which we are not very confident. Finally, we want to make sure that no two features are very close to one another, since this has the effect of adding new low-count constraints.

To do this, we have specified a feature *pattern* language. The feature pattern language is identical to the feature specification language, with the addition of a pattern marker (denoted by the '*' character). After deciding on the feature patterns, they are matched against the training events, generating a set of features in which each pattern marker is replaced by a node label. Each element of the resulting set of features is guaranteed to fire on the training data more than a preset number of times. A single feature pattern may generate multiple features in the presence of a single event[4]. An example of this is given in Figure 12 below. A full list of the patterns we used in the final maximum entropy model is presented in Figure 13.

[4] Making sure that all appropriate features are generated was very tricky. In one instance, a single pattern generated eight features from one event.

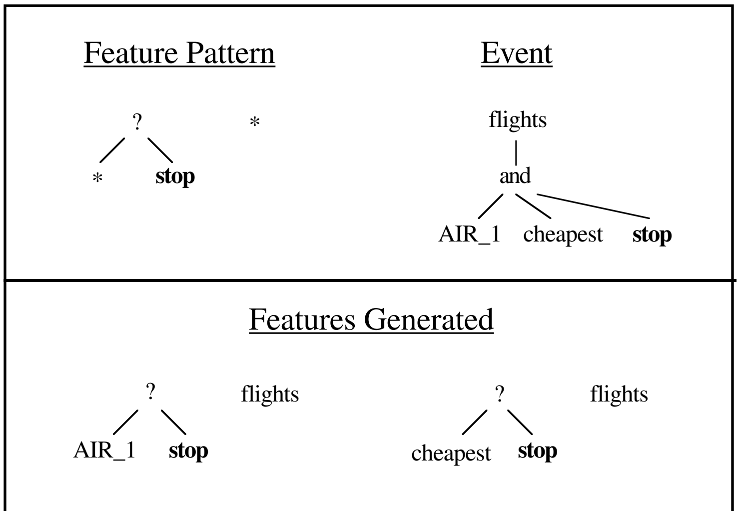


Figure 12. Feature pattern example

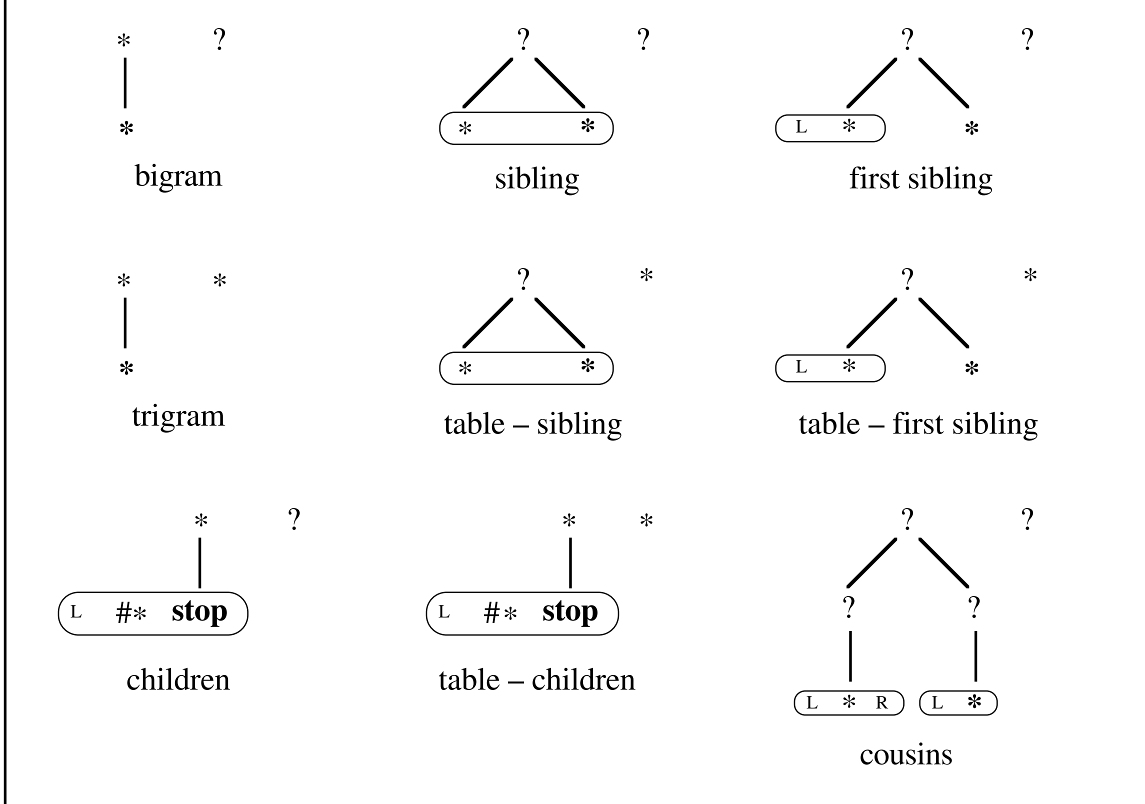


Figure 13. Feature Patterns — Complete List

The *bigram* and *trigram* patterns reinforce the bigram and trigram constraints in the initial distribution. The *sibling* and *table-sibling* patterns are used to collect statistics on the sibling immediately to the left of the future node. This sibling is frequently the node predicted immediately prior to the future node. The *first sibling* patterns are useful because the leftmost child of a node often has special significance. The parse tree grammar tells us that the first node is often either "not", or the most frequently seen label for the current context. The *children* patterns use the '#' syntax to collect statistics about the number of children of each node. This is necessary to accurately predict the *stop* node. Finally, the *cousins* pattern is used to relate the first child of a node to its cousins. The addition of this pattern significantly improved performance, because tagged nodes were difficult to predict accurately. For example, deciding on how probable the label "CITY_2" is depends on whether a node with label "CITY_1" has already been predicted. These labels are often in cousin positions.

5.2.3 Similar Feature Elimination

We have already discussed the need to filter out those features which fire too few times on the training data. We do not want to enforce constraints on our model without first considering the certainty with which the constraints are known. Unfortunately, it is not adequate to use the number of times a feature fires in the training data as the sole criterion for deciding whether or not to keep it. Even if every constraint is derived from a feature that has been observed many times in the data, the interplay between these constraints might implicitly impose additional constraints corresponding to features that are not observed sufficiently [15].

Specifically, if we include two features which fire on exactly the same events with a single exception, we are implicitly including a feature which fires *only* on that exception.

Even if the original features are seen very frequently, the feature which corresponds to their difference will not be seen frequently enough. Our solution to this problem is straightforward. We consider all pairs of features, and compare the set of events which cause each to fire. If the sets differ by less than the preset minimum number, we remove the feature which fires less often. This greedy approach will ensure that every constraint— even those included implicitly — will be observed an adequate number of times in the training data. Unfortunately, this method may throw away more features than is strictly necessary.

5.3 Experimental Results

The feature patterns in Figure 13 generated 1015 features in the feature set, after removing features which fired less than five times in the training data, and eliminating overlapping features. The training data for the maximum entropy model consisted of the same training and heldout data used for the initial distribution. Ideally, this should be yet another unused set of data, but the shortage of complete class A data items (including the NLParse interpretation) has caused us to reuse the data. The training and heldout data together resulted in 63,532 events, while the test data had 7160 events. Using the 4-gram tree-based deleted interpolation model presented in section 4.2 as the initial distribution, the maximum entropy training algorithm was run for 100 iterations, at which point the training entropy was decreasing by only one percent of one percent. The resulting test entropy was 14.1.

Seeing a one bit drop in entropy over the initial distribution was nice, but a little disappointing. At this point we thought that perhaps our initial distribution was just very good. To see the true value of the maximum entropy approach, we reran the experiment using the tree-based *trigram* deleted interpolation model, instead of the 4-gram. The results are given in Figure 14. The relative result was considerably more impressive; the test entropy drops from 21.5 to 14.1. The maximum entropy smoothing brings the entropy down to the same place, regardless of which of the two initial distributions is used.

It is interesting to see the relative progress of the maximum entropy training algorithm. The training entropy reaches half of its eventual improvement in the first two iterations, and ninety percent within the first twenty. Figure 15 is a graph of training entropy versus iterations for the first 25 iterations of the training algorithm for each initial distribution.

	Training	Heldout	Test
Surface NLParse	12.6800	13.4784	15.9474
Flat tree, with parens	18.9160	18.7158	20.3842
Flat tree, without parens	11.5749	12.6873	15.6387
Tree Context Trigram	19.2082	20.1576	21.4704
Tree Context 4-gram	11.7567	13.5918	15.0037
Events	63532		7160
ME, Trigram Initial	11.8296		14.1056
ME, 4-gram Initial	10.8628		14.0814

Figure 14. Comparative Entropies of Different Models

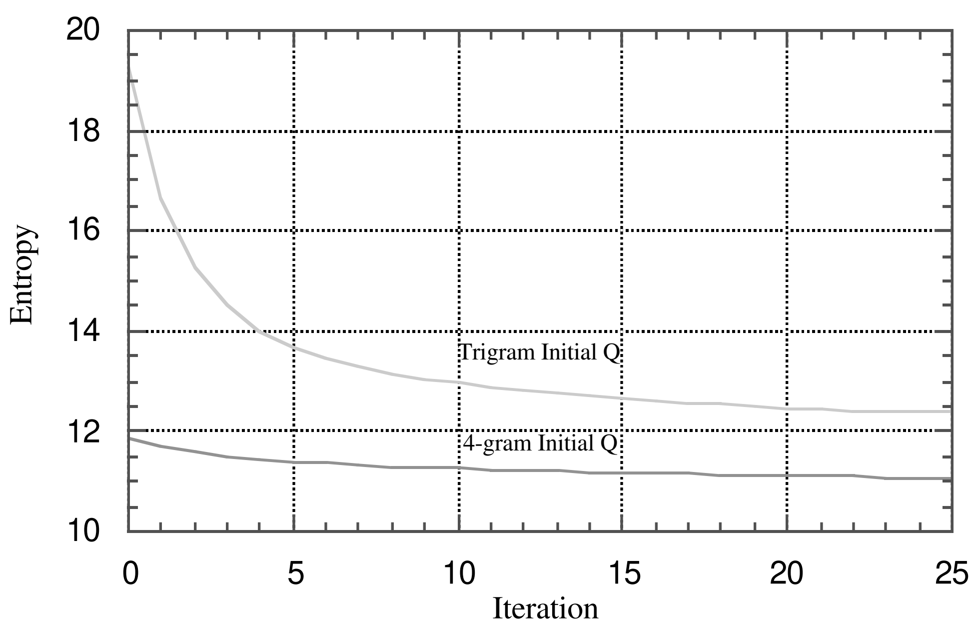


Figure 15. ME Training Entropy

As another way to compare the maximum entropy model with the NLParse trigram model (the first model we built), we randomly generated a set of parse trees from the maximum entropy distribution. 90% of the items generated were valid semantic trees (they could be converted to valid NLParse items), as compared with the 58% of those generated with the trigram model on surface NLParse. In addition, half of the randomly generated items could be found in the original training data.

After doing the original experiment, we experimented with a wide variety of different feature sets (derived from adding or removing feature patterns), and the test entropy did not vary significantly. Lowering the threshold on minimum firing frequency significantly increased the number of features, and resulted in a slight decrease in training entropy, but a marked *increase* in test entropy. This increase can be attributed to overtraining, causing an inability to generalize to new data. If we had another training set, we could have fine-tuned the feature set, but to do so on the test data would invalidate the experiment. By noticing after the fact that changing the features does not significantly affect the model's performance on the test data, we justify using all the training data for training the model, rather than training the features.

6 Frequency Model

After working with the training corpus for a while, we noticed a few peculiarities in the ATIS data. While 874 distinct trees can be found in the 3392 training items, the top thirty most frequently occurring trees account for more than half of the training data! Further, the top seven trees account for a third of the training data, and the top *one* tree accounts for almost 15% — 491 out of 3392 items correspond to the some variation of the question "what are all flights from CITY_1 to CITY_2 ?". These facts weren't so obvious before tagging and putting the data into semantic parse tree form, because the NLParse does not include any kind of canonical ordering. A list of the top seven most frequently seen trees is given in appendix B.

None of the models presented so far explicitly takes the frequencies of entire trees into account. In fact, it seems probable that a language model which assigns every tree observed in the training data a probability proportional to its observed frequency, and every other tree some uniform probability, would do very well indeed. An even better idea would be to smooth the observed training distribution with the maximum entropy distribution found in the previous section. When *T* is the set of semantic parse trees in the training data, and *Ev* is a function which maps trees to events, Equations 6.1, 6.2, and 6.3 define a model *P* on semantic parse trees.

$$F(t) = \frac{c(t)}{|T|} \qquad \text{(Eq 6.1)}$$

$$M(t) = \prod_{x \in Ev(t)} \frac{Q(x)}{Z(x)} e^{\sum_i \lambda_i f_i(x)} \qquad \text{(Eq 6.2)}$$

$$P(t) \;=\; \lambda F(t) \;+\; (1-\lambda)M(t) \qquad \text{(Eq 6.3)}$$

6.1 Frequency Model Entropy

Since we already know a great deal about our training set, and about the maximum entropy distribution, we can predict how well the smoothed model defined by Equation 6.3 is going to do, and use our prediction to test that our implementation is working correctly. First, let *R* be the training set for *F*, *S* be the test set, and define

$$X = S \cap R \qquad\qquad Y = S - R \qquad\qquad n = |X| + |Y|$$

Next, define E_F to be the average (per item) entropy of the training corpus with respect to the distribution *F*, E_P to be the average entropy of the test corpus with respect to *P*, and E_M to be the average entropy of *Y* with respect to *M*. Note in particular that E_M is taken with respect to the difficult (new) items of the test set. We now define E_P as follows:

$$E_P \;=\; -\frac{1}{n}\sum_{t \in S}\log P(t) \;=\; -\frac{1}{n}\sum_{t \in S}\log\left[\lambda F(t) \;+\; (1-\lambda)M(t)\right]$$

By separating *S* into two parts, those items which appear in both the training and test data, and those items which are only seen in the test data (the test items which *F* will assign a zero probability), we can continue as follows:

$$E_P \;=\; -\frac{1}{n}\sum_{t \in X}\log\left[\lambda F(t) \;+\; (1-\lambda)M(t)\right] \;+\; -\frac{1}{n}\sum_{t \in Y}\log\left[(1-\lambda)M(t)\right]$$

At this point we notice that on the items which can be found in the training data, *F* will predict a much higher probability than *M*, so we state the (close) inequality:

$$E_P \;\leq\; -\frac{1}{n}\sum_{t \in X}\log\left[\lambda F(t)\right] \;+\; -\frac{1}{n}\sum_{t \in Y}\log\left[(1-\lambda)M(t)\right]$$

$$E_P \leq -\frac{|X|}{n}\left[\log\lambda + \frac{1}{|X|}\sum_{t\in X}\log F(t)\right] + -\frac{|Y|}{n}\left[\log(1-\lambda) + \frac{1}{|Y|}\sum_{t\in Y}\log M(t)\right]$$

By using the definitions E_M and E_F, we finally arrive at

$$E_P \leq -\frac{|X|}{n}\left[\log\lambda + E_F\right] + -\frac{|Y|}{n}\left[\log(1-\lambda) + E_M\right] \qquad \text{(Eq 6.4)}$$

6.2 Experimental Results

Again, due to lack of available data, we were forced to train both the frequency model and the maximum entropy model on the same data. The frequency model was built from 3166 items, and the two distributions were smoothed together on the remaining 226 heldout items, using the Estimation Maximization algorithm. Ideally we should be able to smooth the two distributions using different heldout data than that which was used to smooth the ME model's initial distribution.

The entropy of the training data is very low; only 7.7 bits per item. This, coupled with the fact that the training data's coverage of the heldout data is high (86%), led us to believe that we would see a dramatic decrease in test entropy using the new model. We hoped that the frequency model component would allow the smoothed model to make accurate predictions on a large percentage of the test data, and that the maximum entropy model would do acceptably on those items not already seen.

The results were surprising: the test entropy had decreased, but by less than half of a bit (as opposed to the two or three bits we expected), to 13.7. Part of the reason was that the training data's coverage of the test data was somewhat lower than expected — only 76% of the 376 test items had been seen in the training data. The main implication, however, was that the maximum entropy model did *very* poorly on items which had never been seen in the training data. In fact, we were able to use Equation 6.4 to get a lower bound on just how poorly the ME model did. Using the experimentally determined values of $\lambda = .507$, $n = 376$, $|X| = 286$, $|Y| = 90$, $E_F = 7.27$, and $E_P = 13.7$, we find that $E_M \geq 30.0$ bits!

When we went back and experimentally determined the entropy of the unseen test items with respect to the maximum entropy distribution, we found that $E_M = 32.1$ bits. All experimentally determined values are given in Figure 16.

Training	Heldout	Test	
Items	3166	226	376
Unseen Items	0	31	90
M on unseen items		31.8182	32.0785
F on seen items	7.67379	6.28285	7.26727
P Entropy	8.11984	10.0565	13.6900

Figure 16. Entropy Statistics of the Frequency Model

7 Conclusions and Future Directions

7.1 Summary and Conclusions

Statistical models have been used successfully in numerous computer applications, including translation, speech recognition, and parsing. Our goal was to see whether standard statistical techniques can be applied effectively to the ATIS task. To that end, we divided the task into five main parts, each of which can be attacked with statistical methods. Figure 17 shows how the parts fit together in the final system. The speech recognition component attempts to generate a ranked list of best text transcriptions for a spoken input. The language model and translation model components are for use by the decoder to decide which intermediate language item is the most probable. Theoretically, the decoder must search through all possible intermediate language sentences to decide which is best. In practice, it uses a variation on the well-known stack decoding algorithm [22] from speech-recognition to prune the search.

The translation model gives the probability of an English text sentence, given any NLParse sentence supplied by the decoder. The language model component produces a probability distribution exclusively on the intermediate language. The "tree-parser" and "tree-unparser" sub-components deterministically translate the NLParse to and from a semantic parse tree format, for use in the language model. In this work, we have explored various statistical language models and analyzed their performance on the ATIS2 task. We have concluded that performance (as measured by perplexity) can be improved by building a language model using semantic parse trees built from the NLParse.

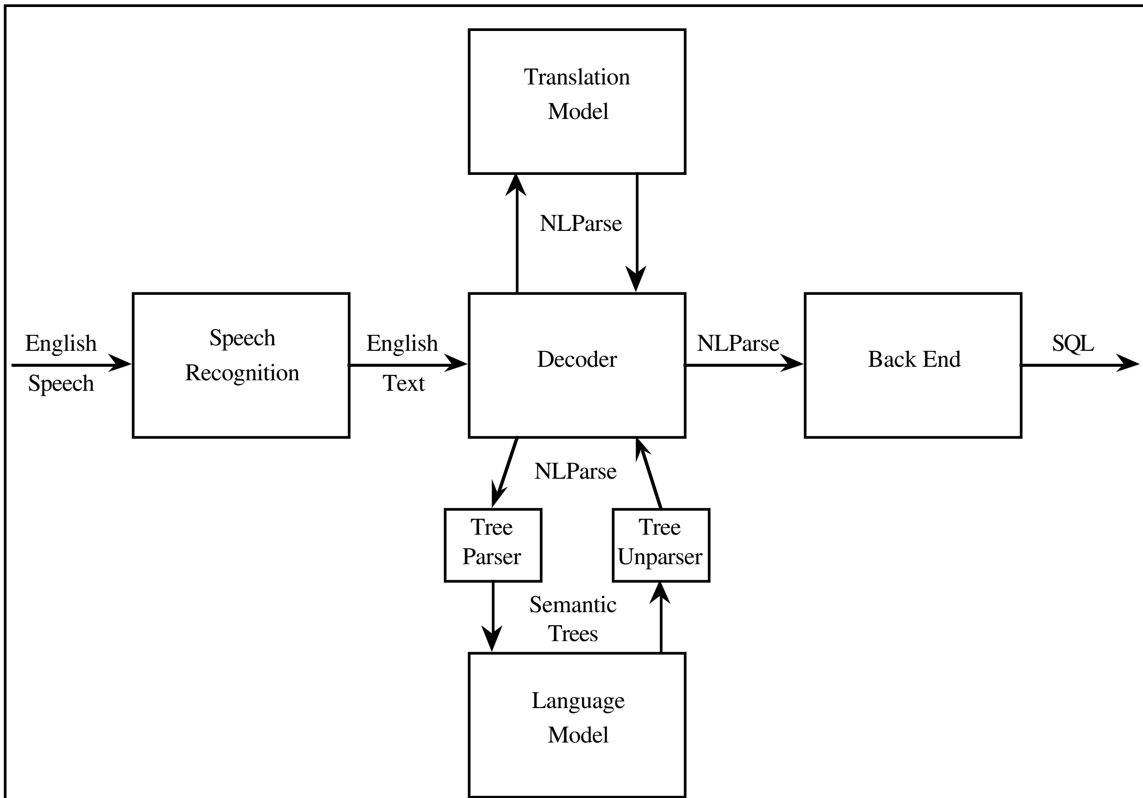


Figure 17. Complete IBM ATIS System

We started by building a deleted interpolation trigram model on the NLParse translations of the English queries. We used a "wall of bricks" bucketing scheme to tie together model parameters associated with low certainty constraints. The test entropy of the NLParse deleted interpolation model was used as the benchmark for comparing the other models built.

After deciding on a general structure for parsing the surface NLParse into semantic trees, we evaluated two trigram models built from the prefix traversal of these trees. The first, which we forced to be able to predict a tree's structure in addition to its content, had a test perplexity which was orders of magnitude worse than the surface NLParse trigram model. The second, which we required only to assign a probability distribution to a tree's content, beat the benchmark perplexity by just 19%. It was clear that a model which understood more about the source language was needed.

The next model we built was a generative deleted interpolation model on the semantic parse trees, with a derivational order corresponding to a prefix traversal of the tree. This one was more clever about where to look for useful statistics on which to train. The context it considered contained the parent of each node, and the table in whose scope that node existed. The model was given considerably more *a priori* knowledge about which trees it could assign zero probability to, allowing it to normalize over a much smaller set. This model's performance was also very poor, which led us to believe that it needed to look at more context for each node. By adding each node's immediate sibling to the available history, the test perplexity was reduced significantly — a 48% decrease from the benchmark.

Next, we introduced the maximum entropy framework for building statistical language models. For the initial distribution of the maximum entropy models, we used the model with the best test performance so far, the four-gram deleted interpolation model on the semantic parse trees. We included features which impose constraints on relevant statistics which can be believed with a reasonable degree of certainty. These include the four-gram, trigram, and bigram constraints used in the initial distribution, in addition to several others which we experimentally determined decrease the training perplexity. The test entropy of this model represented a 72% decrease from the benchmark perplexity, and a 47% decrease from the initial distribution.

We then reran the experiment using the tree-based *trigram* model as the initial distribution. We discovered that after the same number of iterations of the maximum entropy training algorithm, the test entropy was the same as when the four-gram initial distribution was used. Apparently, the maximum entropy model has a lot of tolerance for wide variations in its initial distribution. We suspect, however, that it would not be sufficient to use a uniform initial distribution, because both distributions we *did* use allowed the maximum entropy model to predict a zero probability for certain events. A uniform initial distribution would force the model to predict a non-zero probability for *every* possible event.

Finally, we realized that half of the training data is covered by less than thirty trees. This suggested that if we had a model which was very good at predicting exactly the items in the training data, the chances were good it would do very well on test data. We therefore built a model which assigned an entire item a probability based on its frequency in the training data. Smoothing this model with the maximum entropy model produced a 22% decrease in perplexity over the maximum entropy model alone, which corresponds to a 79% reduction from the benchmark.

As a side product of this experiment, we discovered that the maximum entropy model does terribly on items which it has never seen before. Part of the reason for this is that many of the items exclusive to the test set epitomize the idiosyncrasies of the NLParse grammar. It would be very, very difficult to build a language model which does well on these items. It looks as if this is yet another example of the 80-20 rule; 20% of the effort is spent in achieving 80% of the goal, while 80% of the effort is necessary to go the rest of the way. A comprehensive table containing the performance of all of the language models built in this paper is given in Figure 18.

	Training	Heldout	Test
Items	3000	392	376
Surface NLParse	12.6800	13.4784	15.9474
Flat tree, with parens	18.9160	18.7158	20.3842
Flat tree, without parens	11.5749	12.6873	15.6387
Tree Context Trigram	19.2082	20.1576	21.4704
Tree Context 4-gram	11.7567	13.5918	15.0037
Events	63532		7160
ME, Trigram Initial	11.8296		14.1056
ME, 4-gram Initial	10.8628		14.0814
Items	3166	226	376
Frequency Smoothed	8.11984	10.0565	13.6900

Figure 18. Entropy of All Language Models

7.2 Future Directions

While the maximum entropy framework is very powerful, and it allows us to include a wide variety of information known about a source language, there is a problem with the immense amount of computational power necessary to run either the Generalized Iterative Scaling algorithm [19] or the Improved Iterative Scaling algorithm [20]. More progress needs to be made toward finding fast training algorithms for maximum entropy modelling.

A number of things can be tried to improve the performance of the language models presented in this paper. Enhancing and expanding the feature language might allow the inclusion of important constraints which we did not consider. Allowing the disjunction and conjunction of the current features to create new features is one possible enhancement. Implementing the incremental feature selection proposed by Berger *et al* [15] would probably be a better way of guaranteeing that only good constraints are included in the model. At present, our method of eliminating overlapping features seems a little haphazard.

Another direction for potential improvement is by experimenting with different derivational orders. For example, if we predict how many children a node in the parse tree has before we predict the children themselves (as opposed to predicting the *stop* node at the end), then the model would be able to use that piece of information when predicting the children. It might also be useful to predict all of the children together, rather than one at a time. By predicting all of the children in tandem, a small amount of right context can be used.

Rather than smoothing in the frequency model *after* training the maximum entropy model, it would be more powerful to incorporate constraints from the frequency model into the feature set itself. For example, we could collect statistics on the occurrence of *sub*trees in the training data, and use them in predicting new trees. This might be especially helpful in bringing down the model's entropy on trees which are only seen in the test data. Even if the entire tree has never been seen before, certain parts of the tree may very well have.

Other parts of the overall system might be enhanced by allowing them access to the semantic parse tree format. The system could be rearranged as shown in Figure 19, with the decoder building semantic trees instead of flat NLParse. The translation model could then be modified to report the probability of an alignment between the English and an entire parse tree. The translation model could then more easily take the NLParse structure imposed by its grammar into consideration.

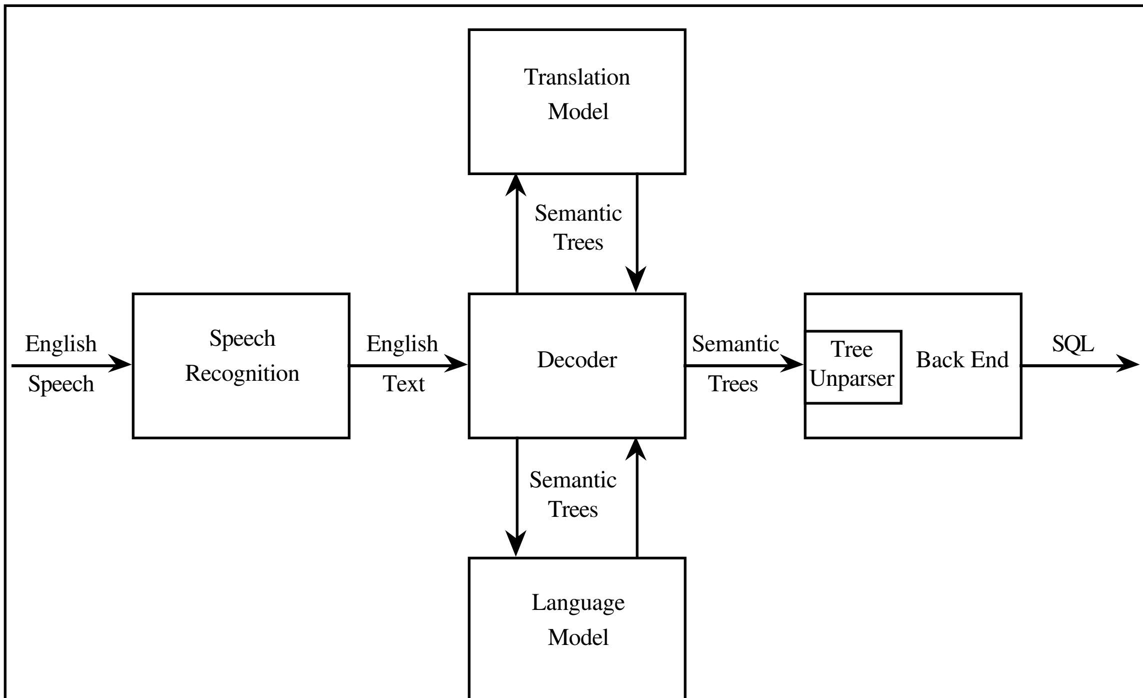


Figure 19. Future IBM ATIS System

References

[1] P. Price, "Evaluation of spoken language systems: The ATIS domain," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Hidden Valley, PA), June 24–27 1990.

[2] M. Bates *et al.*, "Developing an evaluation methodology for spoken language systems," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Hidden Valley, PA) June 24–27 1990.

[3] D. S. Pallett *et al.*, "DARPA February 1992 ATIS benchmark test results," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Harriman, NY), February 23–26 1992.

[4] W. Ward, "The CMU air travel information service: Understanding spontaneous speech," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Hidden Valley, PA) June 24–27 1990.

[5] W. Ward and S. Issar, "Recent improvements in the CMU spoken language understanding system," in *Proceedings of the ARPA Workshop on Human Language Technology*, (Plainsboro, NJ) March 8–11 1994.

[6] J. Glass *et al.*, "The MIT ATIS system: December 1994 progress report," in *Proceedings of the ARPA Spoken Language Technology Workshop*, (Austin, TX), *to appear.*

[7] V. W. Zue *et al.*, "The MIT SUMMIT speech recognition system: a progress report," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Philadelphia, PA), October 15–18 1989.

[8] S. Seneff, "TINA: a natural language system for spoken language applications," in *Computational Linguistics*, vol. 18, no. 1, pp. 61–86, 1992.

[9] R. Pieraccini *et al.*, "Progress report on the Chronus system: ATIS benchmark results," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Harriman, NY) February 23–26 1992.

[10] F. Kubala *et al.*, "BBN BYBLOS and HARC February 1992 ATIS benchmark results," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Harriman, NY) February 23–26 1992.

[11] J. Dowding *et al.*, "Gemini: a natural language system for spoken-language understanding," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Harriman, NY) February 23–26 1992.

[12] A. Berger *et al.*, "The Candide system for machine translation," in *Proceedings of the ARPA Workshop on Human Language Technology*, (Plainsboro, NJ) March 8–11 1994.

[13] F. Jelinek *et al.*, "Decision tree parsing using a hidden derivation model," in *Proceedings of the ARPA Workshop on Human Language Technology*, (Plainsboro, NJ) March 8–11 1994.

[14] E. Vidal *et al.*, "Learning associations between grammars: a new approach to natural language understanding," in *Proceedings of Eurospeech-93*, vol. 2, pp. 1187–1190, (Berlin) September 1993.

[15] L. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," in *IEEE Transactions of Pattern Analysis and Machine Intelligence*, PAMI-5(2): 179–190, 1983.

[16] L. E. Baum, "An Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes", in *Inequalities*, vol. 3, pp. 1–8, 1992.

[17] A. L. Berger, S. A. DellaPietra, V. J. DellaPietra, "A Maximum Entropy Approach to Natural Language Processing," *IBM Technical Disclosure Bulletin*, August 1994.

[18] R. Lau, *Maximum Likelihood Maximum Entropy Trigger Language Model*, S.B. thesis, Massachusetts Institute of Technology, May 1993.

[19] J. N. Darroch, and D. Ratcliff, "Generalized Iterative Scaling for log-linear models," *Annals of Mathematical Statistics*, no. 43, pp. 1470–1480.

[20] S. Della Pietra, and V. Della Pietra, "Statistical modelling using maximum entropy," *IBM Research Report*, in preparation.

[21] J. C. Kowto and P. J. Price, "Data collection and analysis in the air travel planning domain," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Cape Cod, MA), October 15–18 1989.

[22] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The ATIS spoken language system pilot corpus," in *Proceedings of the DARPA Speech and Natural Language Workshop*, (Hidden Valley, PA), June 24–27 1990.

[23] MADCOW, "Multi-Site Data Collection For A Spoken Language Corpus," in *Proceedings of the Darpa Speech and Natural Language Workshop*, (Harriman, NY), February 23–26 1992.

[24] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM Journal of Research and Development*, no. 13, pp. 675–685, November 1969.

Appendix

A Grammar Vocabularies

The following vocabularies are used as the node labels in the semantic parse tree grammar presented in Figure 4.

Feature Operators

all	any	the-maximum	the-minimum	the-number-of

Comparison Operators

equal-to	greater-than	greater-than-or-equal-to	between:comp
contains	less-than	less-than-or-equal-to	

Table Names

aircraft	column-tables	fares	intervals
airlines	compartment-classes	flight-fares	months
airport-services	date-days	flight-legs	restrictions
airports	days	flight-stops	states
cities	dual-carriers	flights	table-tables
class-of-services	equipment-sequences	food-services	time-zones
code-descriptions	fare-bases	ground-services	

Terminal Operators

NULL	containing	from
abbreviated	containing:airport	from:airport
arriving	containing:airports	from:airports
associated-with:class	containing:cities	from:cities
associated-with:	containing:city	from:city
class-of-services	departing	having
associated-with:	equipped-with	having-prices-of
fare-bases	equipping	in:airport
available-for:fare-bases	flying-on	in:airports
available-for:fares	flying-on:days	in:cities
available-on	for	in:city
available-on:days	for:flight-stops	leaving
belonging-to	for:flights	leaving-in
charged-for	found-in	located-in

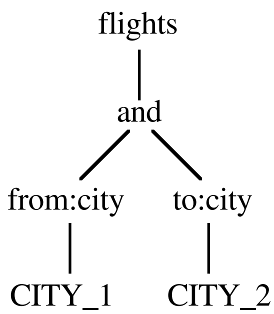
named
named:airline
named:city
named-a-string-containing
numbered
of:class
offered-by
offering
offering:class
on
primarily-served-by
provided-by:airport
provided-by:airports
provided-for:airport
provided-for:airports
provided-for:cities
provided-for:city
provided-with:airport-services
provided-with:ground-services
providing
scheduled-for:flight-stops
scheduled-for:flights
secondarily-served-by
served-by:airport
served-by:airlines
served-by:airports
served-on
serving:cities
serving:city
serving:flights
serving:food-services
serving:meal
stopping-in:airport
stopping-in:airports
stopping-in:cities
stopping-in:city
that-are-legs-for
that-are-stops-for
to
to:airport
to:airports
to:cities
to:city
with-arrivals-on
with-arriving:fares
with-arriving:flights
with-departing:fares
with-departing:flights
with-legs-that-are
with-scheduled

Terminal Names

advance_purchase	cruising_speed	length	service_name
aircraft_code	day_name	low_flight_number	state_code
aircraft_code_sequenc	day_number	main_airline	state_name
aircraft_description	days_code	manufacturer	stop_airport
airline_code	departure_airline	maximum_stay	stop_days
airline_flight	departure_flight_num	meal_code	stop_number
airline_name	departure_time	meal_description	stop_time
airport_code	description	meal_number	stopovers
airport_location	direction	miles_distant	table_description
airport_name	discounted	min_connect_time	table_name
application	dual_airline	minimum_stay	time_elapsed
arrival_airline	dual_carrier	minutes_distant	time_zone_code
arrival_flight_number	economy	month_name	time_zone_name
arrival_time	end_time	month_number	to_airport
basic_type	engines	no_discounts	transport_type
basis_days	fare_airline	note	unit
begin_time	fare_basis_code	one_direction_cost	weight
booking_class	fare_id	pay_load	wide_body
capacity	features	period	wing_span
city_code	flight_days	premium	year
city_name	flight_id	pressurized	AIR_1
class_description	flight_number	propulsion	AIR_2
class_type	from_airport	range_miles	AIR_3
code	ground_fare	rank	AIR_4
column_description	heading	restriction_code	AIR_5
column_name	high_flight_number	round_trip_cost	ARP_1
compartment	hours_from_gmt	round_trip_required	ARP_2
connections	leg_flight	saturday_stay_requird	ARP_3
country_name	leg_number	season	ARP_4

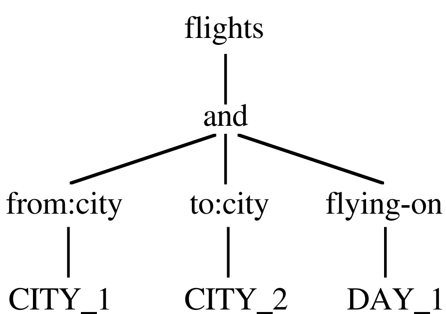
ARP_5
CITY_1
CITY_2
CITY_3
CITY_4
CITY_5
CODE_1
CODE_2
CODE_3
CODE_4
CODE_5
DATE_1
DATE_2
DATE_3
DATE_4
DATE_5
DAY_1
DAY_2
DAY_3
DAY_4
DAY_5
MON_1
MON_2
MON_3
MON_4
MON_5
NUM_1
NUM_2
NUM_3
NUM_4
NUM_5
today-6
today-5
today-4
today-3
today-2
today-1
today+0
today+1
today+2
today+3
today+4
today+5
today+6
a
after
afternoon
airport
am
are
around
associated
at
available
bases
basis
before
belonging
between
breakfast
business_class
by
carriers
charged
cheap
cheapest
class
classes
coach
coach-class
coach-economy_class
code_descriptions
codes
column
columns
compartment_classes
connecting
cost
daily
date
day
dayses
descriptions
dinner
direct
dual
earliest
earliest-arriving
earliest-departing
early
early_afternoon
early_morning
early_evening
economy_class
entries
equipment
equipped
evening
examples
expensive
fare
first
first-class
flight
flight_legs
flying
food
found
ground
help
is
is-a-string
is-between
just
known
largest
late
late_afternoon
late_morning
late_evening
latest
latest-arriving
latest-departing
legs
located
lunch
maximum
mid_afternoon
mid_evening
mid_morning
midday
minimum
minimum_conct_time
modifiers
morning
most
most-expensive
named-a-string
night
no
nonstop
number
number-of
of
offered
one_direction
one_way
overnight
pm
prices
primarily
provided
query
round
round-trip
scheduled
secondarily
sequences
served
service
services
shortest
smallest
snack
stopping
stops
stopses
string
table
tables
than
that
thrift
thrift-class
thrift-economy_class
time
trip
unknown
whose
yes
zones

B Top 7 Most Frequent Trees



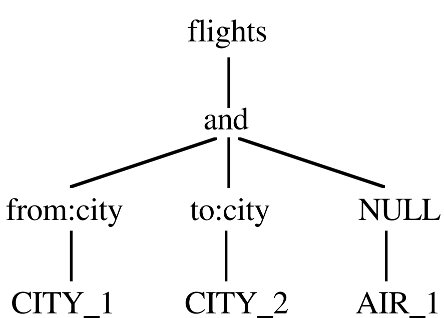
NLParse: list flights from Boston and to Denver

English: What are all flights from Boston to Denver ?



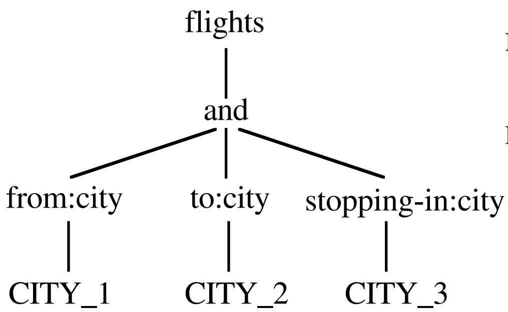
NLParse: list flights from Boston and to Denver and flying on Monday

English: Give me flights that leave on Monday for Denver from Boston.



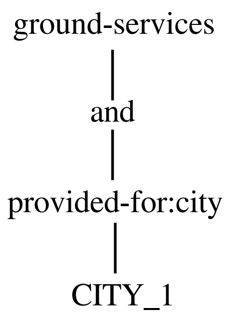
NLParse: list American flights from Boston and to Denver

English: Please tell me which American flights fly to Denver from Boston



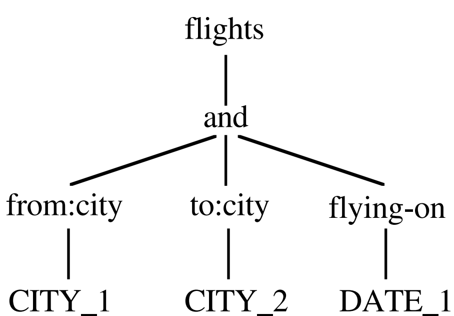
NLParse: list flights from Boston and to San Francisco and stopping in Denver

English: Can I see flights going from Boston to San Francisco with a stop in Denver?



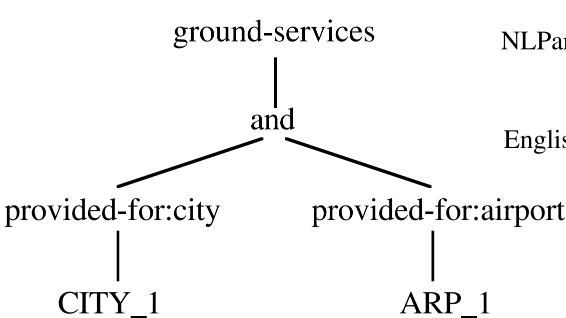
NLParse: list ground services provided for Washington

English: Show me ground transportation in Washington.



NLParse: list flights from Boston and to Dallas and flying on 7/12/91

English: I'd like to fly from Boston to Dallas on August twelfth



NLParse: list ground services provided for BWI and provided for Washington

English: I'm interested in ground transportation between BWI and downtown Washington.