# Probabilistic Techniques for Phrase Extraction*

*Fangfang Feng and W. Bruce Croft*

Center for Intelligent Information Retrieval
Computer Science Department
University of Massachusetts, Amherst

---

**Abstract**

This study proposes a probabilistic model for automatically extracting English noun phrases without part-of-speech tagging or any syntactic analysis. The technique is based on a Markov model, whose initial parameters are estimated by a phrase lookup program with a phrase dictionary, then optimized by a set of Maximum Entropy parameters for a set of morphological features. Using the Viterbi algorithm with the trained Markov model, the program can dynamically extract noun phrases from input text. Experiments show that this technique is of comparable effectiveness with the best existing techniques.

**Keywords:** indexing, phrase extraction, Markov, maximum entropy.
**Additional Keywords:** phrase dictionary, tokenize, smooth, bias

* *Acknowledgments:*
Jay Ponte did the early Markov model design, and provided many suggestions. This material is based on work supported in part by the National Science Foundation under cooperative agreement EEC-9209623. It is also supported in part by United States Patent and Trademark Office and by the Defense Advanced Research Projects Agency/ITO under ARPA order D468, issued by ESC/AXS contract number F19628-95-C-0235. Any opinion, findings and conclusions or recommendations expressed in this material are of the authors and do not necessarily reflect those of the sponsors.

*Address:*
Center for Intelligent Information Retrieval
Computer Science Department
University of Massachusetts
140 Governor St.
Amherst, MA 01003-4610

*Email:*
feng@cs.umass.edu
croft@cs.umass.edu

# 1 Introduction

Extracting noun phrases can be a helpful aid to many applications, such as indexing and summarization. In Information Retrieval (IR), noun phrase indexing can improve the retrieval accuracy and is commonly done in web search engines. To index noun phrases, one must first find a method of identifying or extracting the noun phrases. A straightforward and popular method is to assign a part of speech (POS) tag to every word then collect the sequences of words which correspond to noun phrases (Church, 1988; Smadja, 1993; Brill, 1993; Kupiec, 1993; Voutilainen, 1997; Tolle & Chen, 2000; Perez-Carballo & Strzalkowski, 2000). The POS category of a word reflects its syntactic role in a sentence. The assignment of POS categories to words is referred to as tagging. The tagging problem is difficult because of the lexical ambiguity and the grammatical flexibility. In this approach to noun phrase extraction, a POS tagger plays a major role, and will decide the performance of the overall extraction process.

Tagging methodologies can be categorized into two broad classes, stochastic approaches and heuristic approaches. The stochastic tagger treats the problem as a set of random variables with associated probabilities and uses an overall likelihood to determine the part of speech of a word. The heuristic method uses grammatical and lexical knowledge in the form of a set of rules whose conditions are to be matched against the actual context to deterministically decide the POS of a word. Most stochastic taggers are trained automatically to deal with lexical ambiguities, and many such taggers have incorporated Markov models. A well-trained Markov model can tag sentences more accurately and with more flexibility (Jelinek, 1985; Church, 1988; DeRose, 1988; Cutting, Kupiec, Pedersen, & Sibun, 1992; Kupiec, 1992; Charniak, Hendrickson, Jacobson, & Perkowitz, 1993; Weischedel, Meteer, Schwartz, Ramshaw, & Palmucci, 1993; Xu, Broglio, & Croft, 1994; Merialdo, 1994; Brants, 2000).

Extracting noun phrases from a tagged sentence is still not an easy task, because even after tagging, there can still be considerable uncertainty on issues such as what components or patterns can be noun phrases, how long a noun phrase can be, where a noun phrase should be segmented, etc. For example, we could have a very simple pattern for the noun phrase extraction, an adjective followed by a proper noun. Here are two sentences:

*They were due Sunday.*
*They were young Americans.*

They will be tagged exactly same as "PPSS BED JJ NP" (the tag annotations are defined in Brown Corpus). However, *"due Sunday"* of the first sentence is not a noun phrase while *"young Americans"* is. If we specify that the name of a day cannot be a component of a noun phrase, then this excludes other possible phrases such as *"bloody Sunday"*. There are many such conflicts between numbers, months, years, and past, passive, and present participle verbs.

The length of a noun phrase is another issue. The longer the noun phrase (e.g. number of words $> 3$), the more modifiers and the more specific it is, while shorter noun phrases tend to be more general. Shorter noun phrases usually have higher frequency than phrases preceded by more modifiers (i.e. longer ones). For example, in the 1987 Wall Street Journal from the TREC collection (Harman, 1995), there is a long noun phrase *"IBM version of new operating system software"*. This phrase occurs only once while *"operating system software"* occurs 22 times and *"operating system"* occurs 211 times. Some IR systems perform more effectively when short phrases are identified in the queries. This raises the problem of how to segment a long phrase into shorter ones. For example phrase mentioned above can be segmented into *"IBM version"* and *"new operating system software"* or *"operating system software"*, or *"operating system"*, or *"system software"*. It is difficult to decide which segmentation is better without more semantic information. These examples show that even if a POS tagger can supply accurately tagged text, a noun phrase extractor still needs to solve a variety of other problems.

A method of noun phrase extraction that does not require syntactic tags is to use delimiters such as verbs and stopwords to tokenize a sequence of words between delimiter boundaries. Such a method can not avoid

the problem of lexical ambiguity. For example, the word *"can"* is not only in a typical stopword list but also a noun. The word *"fell"* can be a noun, a verb, or an adjective. It would be difficult to extract the noun phrase *"beverage can"* from the sentence, *"The beverage can fell"*, even with a collection of heuristic rules for confusing cases. Using only lexical or morphological knowledge to identify the delimiters is often not sufficient. A "surface grammatical analysis" can be helpful for improving the performance of this kind of method (Bourigault, 1992).

For IR applications, the desired characteristics for a phrase extraction system are that it is fast, accurate, trainable, has low storage overhead (i.e. no large dictionaries), and can recognize phrases that have not previously been encountered. None of the current techniques has all of these characteristics. In this paper, we investigate an approach that combines a simple tokenizing method with the method of automatically training a Markov model for extracting noun phrases.

In the next section, we describe previous research related to phrase extraction. In Section 3, we describe our approach and Markov model in more detail. Section 4 shows how the model is initialized and trained. Section 5 presents a series of experiments and discusses the results. Conclusions and future research are covered in Section 6.

## 2 Previous Work

This section describes previous work related to noun phrase extraction. As we described in the previous section, most extraction systems employ a POS tagger. Church proposed a stochastic method (Church, 1988) to extract noun phrases based on statistical information. The method counts the frequencies for POS tags of every word and every trigram from the Brown Corpus, and also estimates the probabilities of noun phrase boundaries from tagged training material (about 40,000 words and 11,000 noun phrases). The first pass associates every word with a POS tag. The second pass uses precedence parsing to insert the brackets around noun phrases. The reported performance was very good, accuracy of 95-99% (recall was not reported).

LEXTER (Bourigault, 1992) does "surface grammatical analysis" which uses delimiters, such as verbs, stop words, punctuation, premodifiers, nominal heads, and certain kinds of postmodifying prepositional phrases and adjectives, to tokenize French texts for extracting *maximal length noun phrases*. LEXTER can extract most terminological noun phrases. The effectiveness was measured to be 95% recall, but, precision was not reported.

XTRACT (Smadja, 1993) uses straight statistical measures to retrieve from a corpus (10 million-word corpus of stock market news reports) pairwise lexical relations whose common appearance within a single sentence are correlated. For those pair (bigram, any order) whose frequency of occurrence is above a certain threshold and if words are used in relatively fixed ways, XTRACT produces collocations involving more than two words (ngrams (Choueka, 1988)). For example, the bigram "average industrial" produces the ngram "the Dow Jones industrial average", since the words are always used within the same noun phrases in the training corpus. XTRACT uses a parser Cass (Abney, 1990) to add syntactical information to collocations (bigrams) and filters out inappropriate ones as well as such ngrams containing them. For example, a bigram "price rose" is identified as subject-verb by Cass, then filtered out and all ngrams containing it. After the filtering, the remaining ngram ($n > 1$) is considered as a noun phrase. The effectiveness of this system was measured at 94% recall and 80% precision. The technique is very similar to the "statistical phrases" used in early IR research by Salton and Lesk (Salton & Lesk, 1968) and studied extensively by Fagan (Fagan, 1989).

NPtool (Voutilainen, 1997) uses a lemmatizer, ENGTWOL, to analyze English text morphologically, and assign a ENGTWOL-style description for each word. Then uses a constraint grammar parser to parse the text and extract noun phrases. If an ambiguity is found by the parser, the parsed text will be processed by two different finite-state parsers, NP-hostile and NP-friendly with different NP-hood heuristics, then the intersections of noun phrases of two parsings will be extracted. The reported effectiveness was 98.5-100%

recall and 95-98% precision.

Kupiec uses a noun phrase extractor in a question-answer system MURAX (Kupiec, 1993). The NP extractor employs a POS tagger which is based on a hidden Markov model and trained on the untagged words of half of the Brown Corpus (Francis & Kučera, 1982), it identifies noun phrases by finite-state recognizers for matching lexicon-syntactic patterns.

AZ Noun Phraser was developed by University of Arizona (Tolle & Chen, 2000), and combines tokenizing the text with POS tagging of tokens (Brill, 1993). It then applies a set of grammatical rules against the tagged tokens to identify noun phrases. In a more general sense, the noun phrase is also considered concept. There are also approaches for identifying other type of phrases, such as (Ratnaparkhi, Reynar, & Roukos, 1994; Ratnaparkhi, 1998) using a maximum entropy model for prepositional phrases, and (Haase, 1996) using a FramerD for all types of phrases.

Compared with the above, the approach described in this paper is a mixed method but uses no grammatical or syntactical knowledge. A Markov model is trained to recognize phrases directly rather than POS tags. The development of the new extraction system was undertaken because of the problems with existing techniques. Despite the high reported effectiveness of the extraction systems based on POS tags, in an IR setting they were found to make a number of serious errors and not able to easily adapt to short queries and new vocabulary. The techniques based on statistical phrases were more flexible in that they capture the phrases used in the corpora being searched, but this technique requires large dictionaries and a number of heuristics to attain reasonable levels of effectiveness. The combination of techniques used in the extraction system described here was designed to overcome these problems.

# 3 The Phrase Extraction Approach

Our approach applies a combination of methods to extract noun phrases. First, a tokenizer generates tokens (i.e. a sequence of words, also called the phrase candidates) from free text using a list of delimiters and a set of delimiting rules, then a trained Markov model extracts the noun phrases from the tokens. The tokens are called phrase candidates because they may or may not contain subsequences of words considered noun phrases. To recognize phrases from the tokens, there are two different phases, a training phase and an application phase. During the training phase, the training program looks for every token subsequence of two-or-more words in a phrase dictionary. After the Markov model has been trained, during the application phase, a dynamic program with the Viterbi algorithm (Viterbi, 1967) recognizes phrases from the token by finding the best subsequence with the maximum likelihood through the model. In this section, we describe the tokenizer and define the Markov model.

## 3.1 Tokenizer

The tokenizer is a case-sensitive scanner, which generates tokens by identifying delimiters from text. There are two types of delimiters, boundary delimiters and removal delimiters. The boundary delimiters are a set of special words. Some of them can be used for starting a new token, for example, personal titles such as *Mr.* and *Prof.* Some of them can be used for terminating a token, for example, company designators such as *Corp.*, *Lit.*, and *A.G.*

The removal delimiters can be any items which are considered not significant for any noun phrases, and will be discarded by the tokenizer. There are five types of removal delimiters:

1. stopwords: articles, pronouns, prepositions (excluding *"of"*), conjunctions, some adverbs (e.g. what, when, where, and so forth), number words (e.g. one, ten, hundred), unambiguous verbs that can be used as

verb only (e.g. be, expect), those past and passive participles of most frequently used irregular verbs (e.g. known, bought, got), and words about times (e.g. minute, hour, week); stopwords do not include those verbs that can be nouns (e.g. can, make).

2. numbers: numeric items, such as currency, percentage, and fractions, are removed.
3. punctuation: only few exceptions, like hyphen used in compound words, quotes used in possessives or as an *and* (e.g. *rock 'n' roll*), and periods used in abbreviations, most others are considered removal delimiters.
4. verb patterns: there are about ten verb patterns used by the tokenizer to predicate verbs and remove them such as, *have to* {verb}, *able to* {verb}, and *ought to* {verb}.
5. formating delimiters: such as table fields, labeled lines, and section heads are considered removal delimiters.

The tokenizer parses each text item (single word). If the item is not a delimiter (any type) then it will be saved in a buffer. If it is a boundary delimiter, the tokenizer will starts or terminates a token. To start a new token, the tokenizer outputs the saved items in the buffer and puts the boundary delimiter in the buffer as the first word. To terminate a token, the tokenizer puts the boundary delimiter at the end of the buffer and outputs the buffer. If the item is a removal delimiter, it will be discarded and the tokenizer will output the buffer.

Here we assume that the text is written in a normal format, like a newspaper article, that capitalizes proper names and the first word of a sentence. For such a text the tokenizer can recognize the proper names and do a simple sentence segmentation based on these capitalized words. There are few exceptions where the stopwords, numbers, or certain punctuation used by proper names are not treated as the removal delimiters. Table 1 shows examples of such items used in proper names. The proper name, here, is defined as uninterrupted of sequence of capitalized words and special items (e.g. de, von, las, &, "R") in certain pattern. The tokenizer uses few rules to protect removing such items from the proper names by the pattern matching. If the item occurence matches the rule it will be saved in the buffer, otherwise, like other removal delimiters, it will be discarded.

| proper name | delimiter |
|---|---|
| World War I | I |
| Pan Am | Am |
| American Can | Can |
| U.S. Today | Today |
| vitamin A | A |
| AT&T | AT,& |
| Toys "R" Us | ", Us |
| Six Flags | Six |
| Red October | October |
| S & P 500 index | &,500 |
| Windows 3.1 | 3.1 |

Table 1: Removal delimiters used in proper names

As an example of how the tokenizer works, if the input text is:

> It disclosed that its 28.1%-owned A&W Brands Inc. affiliate filed with the Securities and Exchange Commission for an initial public offering of 3.4 million shares, with 2.3 million to be sold by the company and the rest by holders, at $13 to $15 a share, through underwriters led by First Boston Corp. Proceeds will be used to pay bank debt.

the output tokens are:

A & W Brands Inc
affiliate filed
Exchange Commission
initial public offering
First Boston Corp
bank debt

Note that a token could contain not only words but also other items such as numbers and ampersands. In the rest of paper, however, we refer to the items of a token simply as *"word"*.

### 3.2 Markov Model

As mentioned earlier, our approach can be classified as a Markov model. A Markov model is any probabilistic process in which the future development is completely determined by the present state and not at all by the way in which the present state arose (Baum & Eagon, 1963). It can be defined as $(A, B, \pi)$, where $A$ is a set of probabilities of state-to-state transitions; $B$ is a set of probabilities of the process generating a symbol at a certain state; $\pi$ is a set of probabilities of initial states (Rabiner & Juang, 1986). In our Markov model, a set of $n$ states represents word positions of a phrase $S = \{s_0, s_1, \ldots, s_n\}$, where $n$ is the maximum length of a phrase. State 0, $s_0$ indicates no phrase or initial state, $s_1$ through $s_n$ indicate the *1st* through *nth* word position of a phrase, and $s_n$ could also indicate a terminal state ending of a phrase (Figure 1.b). For a first-order model (i.e. a single word of context), we may define $S_r$ and $S_{r-1}$ as random variables denoting the states of the word at any position $r$ and the proceeding word at the position *r-1* in a phrase. The transition probability $a_{ij}$ linking two states $s_i$ and $s_j$, represents the probability of state $s_j$ following $s_i$, i.e. $p(S_r = s_j | S_{r-1} = s_i)$.

The word at position $r$ is represented by a random variable $V_r$, which ranges over the vocabulary $V = \{v_1, \ldots, v_z\}$, where $z$ is the number of words (i.e. the size of a vocabulary including all allowed terms). Discrete state-dependent word probabilities $b_{jk}$ represent the probability that word $v_k$ is seen at the state $s_j$ (in our case, $v_k$ occurs as the *jth* word of a phrase), i.e. $p(V_r = v_k | S_r = s_j)$.

Given that $t$ represents a clock time, a random variable $\Pi_t$ is defined to indicate the current state at time $t$, so that the probability of initial state $\pi_i$ represents the probability of a sequence beginning with state $s_i$, i.e. $p(\Pi_t = s_i | t = 1)$. We now formally define the Markov model $(A, B, \pi)$ as:

$$\begin{aligned} A &= \{a_{ij} | \quad 0 \leq i, j \leq n\} \\ B &= \{b_{jk} | \quad 0 \leq j \leq n, 1 \leq k \leq z\} \\ \pi &= \{\pi_i | \quad 0 \leq i \leq n\} \end{aligned}$$

Our approach is described by a non-ergodic model where we impose constraints on the phrase length, state transition and initial states. The maximum phrase length is six (i.e. $n = 6$). The transitions represent the word positions of a phrase (i.e. nonzero states, $s_1$ through $s_6$) only allow from a word to the next word and the initial states only allow $s_0$ and $s_1$ (i.e. non-phrase state and the state having seen the first word of a phrase), so that some of transitions in $a_{ij}$ and most of $\pi_i$ values are zero.

The model stays at $s_0$ if there no phrase occurs in the input tokens. For the model to recognize a phrase, it must see the first word of a phrase, then the second, and so forth. The state goes from $s_0$ to $s_1$, and from $s_1$ to $s_2$ for the second word, and so on for the rest of the phrase words until the end of the phrase. When the model has seen the last word of the phrase the state could either directly shift back to the initial state or go to a terminal state then return to the initial state. We have tried both ways, referred to as $model_a$ and $model_b$ and shown as Figure 1.a and Figure 1.b. $model_a$ has no common terminal state. In such a model every non-initial state could be a terminal state. $model_b$ has a common terminal state, and whenever an ending word of a phrase is seen, the model will go to the common terminal state $s_6$ then return to the initial state.

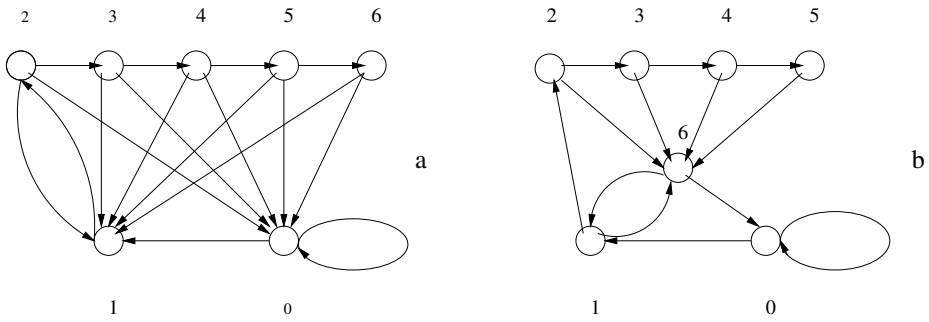


Figure 1: Two examples of non-ergodic model

We tried the $model_a$ in our initial experiments, but we found that many extracted phrases ended with an incorrect word (i.e. not a noun). $model_b$ uses the common terminal state $s_6$ to distinguish this state from other states, specially to distinguish nouns from other type of words, such as adverbs, adjectives, and past participles. For example,

organized crime
armed force

The words *organized* and *armed* can never be used as nouns so that their probabilities of occurring at the end of a phrase will be zero (i.e. $b_{6i} = 0$). State $s_6$ is not redundant as a terminal state of other nonzero states, but rather it is an important feature for head nouns and is used for optimizing and smoothing (see Section 4.4 and 4.5).

In $model_b$, $\pi_i$ has a nonzero probability only when the initial state is either $s_0$ or $s_1$ (seeing no phrase or seeing a phrase). Correspondingly the initial transition $a_{0j}$ could have nonzero probability only when $j = 1$ or $j = 0$ (starting a phrase or staying in the non-phrase state). The other transition probabilities $\{a_{ij}|0 < i, j \leq 6\}$ will have nonzero value either when going to the next state (i.e. $j = i + 1$), that is, going from a word to the next word of a phrase, or the terminal state (i.e. $j = 6$). Since the extractor is interested only in phrases of length two or more, the transition from the first word to the second word of a phrase is always available. The transition could go from $s_1$ to $s_6$ with $a_{16}$ for the two-word phrase or to $s_2$ with $a_{12}$ for a longer phrase.

The transition probabilities from terminal state $s_6$ are $a_{6j}$ and will have a nonzero value only when going to $s_0$ or to $s_1$ (i.e. $a_{60} > 0$ and $a_{61} > 0$). Going to $s_0$ indicates either going from an end of a phrase to the non-phrase, or from the end of the phrase which is also the end of a token to the initial state waiting for the next token. Going to $s_1$ indicates going from the end of a phrase to the beginning of another phrase.

# 4 Training

Once the model is defined, the parameters $(A, B, \pi)$ can be refined by a training procedure of three phases. Since we are using a supervised training method, the parameters could be estimated by a training corpus with bracketed noun phrases, or phrase dictionary lookup. Because there are no training corpus available to us, we build a phrase dictionary in the first phase. The second phase is calculating initial estimates of the parameters with phrase dictionary lookup. The final phase involves optimizing the parameters using the Maximum Entropy Principle (Jaynes, 1957a; Jaynes, 1957b).

## 4.1 Building Phrase Dictionary

Since there was no large, publicly-available phrase dictionary, building a phrase dictionary was an important first step in the training process. The total number of phrase entries in two popular dictionaries available on-line (Collins English Dictionary, 1979 edition and Longman Dictionary of Contemporary English, 1978 edition) is 19,823. Although this could be a large number for some applications, it is not sufficient for training the Markov model for phrase extraction. Instead, we used a four-step heuristic algorithm to collect phrases from all TREC collections (6.9 gigbytes of text):

1 Insert tokens in a hash table.

2 Remove non-noun words from the end of tokens by rules or other heuristics (see Section 5.1).

3 Cluster tokens whose occurrence frequency is lower than a threshold.

4 Discard tokens whose occurrence frequency is lower than a cutoff value.

The program gets tokens from the tokenizer and hashes them into a hash table, counting the frequency for every entry. After all tokens have been inserted in the hash table, in step 2, the program invokes a set of simple rules to recursively remove obvious non-noun words from the end of entries in the hash table. Here are two examples of simple rules:

1 **if** the three-letter suffix of the word has never been used in any noun **then** remove it;

2 **if** the two-letter suffix of the previous word is *"ly"* **and** the suffix of the ending word is any of *"ed", "ing", "ble" "ly"* **then** remove both of them;

A table of three-letter suffixes of nouns are available to the rule 1. This information was collected from Collins English Dictionary (1979 Edition) which includes 81,958 words and 29,494 nouns. To recognize those plural suffixes of nouns, the program does a simple stemming for words ending with *s*, *es*, and *ies*, assuming the suffix cannot be found in the three-letter suffixes table. After each removal, the original entry is deleted from the hash table and the remainder of the token is inserted if it still has two or more words. For the rest of entries in the hash table, the program uses the average frequency as a threshold. From the TREC collections, there were 72,162,082 tokens and 4,114,697 entries total so that $threshold = 17.5376$.

A clustering procedure (step 3) tries to replace low frequency entries with high frequency entries. If any subsequence of a low frequency entry is identical to a high frequency entry, then the procedure removes the low frequency entry and increases the count of the high frequency entry by the count of low frequency entry. For example, Table 2 shows part of a hash table before the clustering and Table 3 shows the result of clustering.

| freq | tokens |
|---|---|
| 1 | 3-D workstation industry matures |
| 41 | workstation industry |
| 26 | 3-D workstation |
| 8 | industry matures |

Table 2: Example of the hash table of tokens before clustering

*"3-D workstation industry matures"* is combined with each other three entries so it is replaced. *"industry matures"* is not replaced even though its frequency is below the threshold, because it does not contain any subsequence that matches a high frequency entry. Thus, after the clustering there can still be some low

| freq | tokens |
|---|---|
| 42 | workstation industry |
| 27 | 3-D workstation |
| 9 | industry matures |

Table 3: Example of the hash table of tokens after clustering

frequency entries left in the hash table. Those entries of frequency less than the cutoff value of 3 are discarded. Usually the cutoff value is less than the threshold, but there is no known way to choose the cutoff value automatically and optimally (see Section 5.1). The phrase dictionary is composed of the remained entries from the cutoff, and will be used in the training phases.

## 4.2 Initialization

Once the phrase dictionary has been built, the initial model parameters $(A, B, \pi)$ can be estimated. The initial estimates are obtained by doing phrase lookups of the training data. The lookup procedure attempts to find any subsequence of the training tokens from the dictionary (the "full match") from the longest (six words) subsequence to the shortest (two words). Thus the full match considers every possible subsequence as a training sequence. The states, transitions and words are counted along the way using three arrays. The state array s[i] records the total times of $s_i$ has been accessed; the transition array a[i,j] records how many times a transition has been made from $s_i$ to $s_j$, and the word array v[k,j] records the times of that word $v_k$ has been seen at state $s_j$. From these arrays, the initial probability (or the observed probability) estimates can be obtained.

The 1987 Wall Street Journal collection (131.7 MB) was used as the training data. It includes 46,448 documents, 2,349,946 tokens, and 6,125,944 words, where 165,355 are unique. Using the previous example *"IBM version of new operating system software"*, as the single training token, the subsequence lookup procedure and the status of state array s[i] are showed in Table 4.

| subsequence | s[0] | s[1] | s[2] | s[3] | s[4] | s[5] | s[6] |
|---|---|---|---|---|---|---|---|
| IBM version of new operating system | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| version of new operating system software | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| IBM version of new operating | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| version of new operating system | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| IBM version of new | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| version of new operating | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| new operating system software | 6 | 1 | 1 | 1 | 0 | 0 | 1 |
| version of new | 7 | 1 | 1 | 1 | 0 | 0 | 1 |
| operating system software | 7 | 2 | 2 | 1 | 0 | 0 | 2 |
| new operating system | 7 | 3 | 3 | 1 | 0 | 0 | 3 |
| IBM version | 7 | 4 | 3 | 1 | 0 | 0 | 4 |
| new operating | 7 | 5 | 3 | 1 | 0 | 0 | 5 |
| system software | 7 | 6 | 3 | 1 | 0 | 0 | 6 |
| operating system | 7 | 7 | 3 | 1 | 0 | 0 | 7 |

Table 4: Example of the state array

For each subsequence of adjacent words in the example token, Table 4 lists accumulated counts for every state that is accessed. For example, when the lookup procedure finds *"new operating"* in the phrase dictionary, it means that word *"new"* has been seen at state one (i.e. the first word of the phrase) and the word *"operating"* has been seen as the end of the phrase (i.e. see the word at the terminal state), so $s_1$ and $s_6$ will increase by one. The transition array a[i,j] and word array v[j,k] are updated correspondingly, where $j$ is the state. Assume

that the word identifiers of *"new"* and *"operating"* are 19 and 20 respectively. When *"new operating"* is found in the dictionary, the elements v[1,19] and v[6,20] of the word array, and a[0,1] and a[1,6] of the transition array will all increase by one. After seeing the end of the phrase, the state transition will either return to the non-phrase state $s_0$ or go to the state of another phrase start (i.e. $s_1$), depending on the dictionary lookup for the next subsequence. If the next subsequence *"system software"*, is also found, then a[6,1] will be increased by one, otherwise one is added to a[6,0]. This process is applied to every subsequence of all training tokens. After this process, we can get the initial parameter estimates from the arrays:

$$\begin{aligned} \pi_0 &= s[0]/(s[0]+s[1]) \\ \pi_1 &= s[1]/(s[0]+s[1]) \\ a_{ij} &= a[i,j]/s[i] \\ b_{jk} &= v[j,k]/s[j] \end{aligned}$$

The initial model then can be used to extract phrases via dynamic programming. For a given word sequence (token), the Viterbi algorithm (Viterbi, 1967) is used to find the best path (state sequence) with the maximum likelihood through the model. For the above example, the best path is (1 6 0 1 6 0 1) through the initial model, so the extracted phrases are *"IBM version"* and *"new operating"*. This example indicates that the initial model needs to be optimized since it missed an important phrase *"operating system"*. After the optimization, the extracted phrases from this example will be *"IBM version"* and *"new operating system software"*.

## 4.3 Optimizing with Maximum Entropy

The parameters $(A, B, \pi)$ of a Markov model can be adjusted in variety of ways. For example, a simple method is to bias $A$ and $B$ (see Section 5.3). The most well-known method is Baum-Welch algorithm (Baum & Eagon, 1972), which is an iterative procedure that maximizes the probability of the observation sequence given to the model. Our approach is to find a set of biases for different *word clusters*. In a word cluster every word has the same features (see Section 4.4). A Maximum Entropy (ME) model (Jaynes, 1957a; Jaynes, 1957b) is well-suited for such an approach since it combines diverse forms of contextual information in a principled manner, and does not impose any distributional assumptions on the training data.

The probability model is defined over $S \times V$ described in the Section 3.2. The probability of seeing a word $v$ at state $s$ is defined as:

$$p(s|v) = \frac{1}{Z(V)} \cdot \prod_{j=1}^{k} \alpha_j^{f_j(s,v)} \tag{1}$$

$$Z(V) = \sum_{v \in V} \prod_{j=1}^{k} \alpha_j^{f_j(s,v)}$$

where $Z(V)$ is a normalization factor to ensure $\sum_v p(s|v) = 1$, while $\alpha_1, \ldots, \alpha_k$ are the positive model parameters, $f_1, \ldots, f_k$ are known as *features*, and $f_j(s,v) \in \{0,1\}$. Note that each parameter $\alpha_j$ corresponds to a feature $f_j$, and can be interpreted as a *weight* of the feature. The probability $p(s|v)$ is a normalized product of those features that are *active* on the pair $(s,v)$, i.e. those features $f_j$ such that $f_j(s,v) = 1$. The parameters $\alpha_1, \ldots, \alpha_k$ of the probability distribution $p^\star$ that best fit the training data can be obtained with the maximum likelihood estimation (Ratnaparkhi, Reynar, & Roukos, 1994):

$$Q = \{p | p(s|v) = \frac{1}{Z(V)} \cdot \prod_{j=1}^{k} \alpha_j^{f_j(s,v)}\}$$

$$L(p) = \sum_{s,v} \tilde{p}(s,v) \log p(s|v)$$

$$p^\star = arg \max_{p \in Q} [L(p)]$$

where $Q$ is the set of models of log-linear form; $\tilde{p}(s,v)$ is the probability of seeing $v$ at state $s$ in the training; $L(p)$ is the conditional log-likelihood of the training set; and $p^\star$ is the optimal probability distribution according to the maximum likelihood criterion.

Finding the $p^\star$ is implemented as constraints on the model $p$'s expectation of features of $f_j$, the constrains are given by:

$$E_p f_j = E_{\tilde{p}} f_j \qquad 1 \leq j \leq k \tag{2}$$

where the model's feature expectation is (Lau, Rosenfeld, & Roukos, 1993):

$$E_p f_j = \sum_{v \in V, s \in S} \tilde{p}(v) p(s|v) f_j(s,v)$$

and the observed feature expectation is:

$$E_{\tilde{p}} f_j = \sum_{v \in V, s \in S} \tilde{p}(s,v) f_j(s,v)$$

$\tilde{p}(s,v)$ indicates an observed probability of $(s,v)$, and $\tilde{p}(v)$, an observed probability of word $v$ in the training data. Thus, the constraints compel the model to match its feature expectations with those observed in the training data.

This model also can be interpreted under the Maximum Entropy. The entropy of the distribution $p$ is defined as:

$$H(p') = - \sum_{s \in S, v \in V} p(s,v) \log p(s,v) \tag{3}$$

$$P = \{p' \mid E_{p'} f_j = E_{\tilde{p}} f_j, \quad j = 1, ..., k\}$$

The maximum likelihood parameter estimation for the models of form (1) being equivalent to the ME parameter estimation models (3) (Darroch & Ratcliff, 1972). That is

$$p^\star = arg \max_{p \in Q} L(p) = arg \max_{p' \in P} H(p')$$

if $p$ has the form (1) and satisfies the $k$ constraints of (2).

We estimate the parameters $\alpha_j$ with the algorithm of Generalized Iterative Scaling (GIS) (Darroch & Ratcliff, 1972):

$$\begin{aligned} {\alpha_j}^{(0)} &= 1 \\ {\alpha_j}^{(m+1)} &= {\alpha_j}^{(m)} (\frac{E_{\tilde{p}} f_j}{E_{p^{(m)}} f_j})^{\frac{1}{C}} \end{aligned}$$

where $C = \max_{s \in S, v \in V} (\sum_{j=1}^{k} f_j(s,v))$ and $E_{\tilde{p}} f_j$ are computed from the initial training for the given $k$ features. The GIS procedure re-estimates the model's expectation $E_p f_j$ literately for $j = 1, ..., k$. The computation of $E_p f_j$ involves summing over each $v \in V$ and $s \in S$, such that:

$$
\begin{aligned}
E_{p^{(m)} f_j} &= \sum_{s,v} \tilde{p}(v) p^{(m)}(s|v) f_j(s,v) \\
p^{(m)}(s|v) &= \frac{1}{Z(V)} \prod_{j=1}^{k} \alpha_j^{(m)^{f_j(s,v)}}
\end{aligned}
$$

Each iteration will get a $L(p)$ and $L(p^{(m+1)}) \geq L(p^{(m)})$. After enough number of iterations $p^{(m)}$ will converge to $p^\star$, that is $\lim_{m \to \infty} p^{(m)} = p^\star$ (Darroch & Ratcliff, 1972), and a set of parameters $\alpha_{1,\ldots,k}$ for features $f_{1,\ldots,k}$ will become stable.

## 4.4 Features for Word Cluster

The conditional probability $p(s|v)$ is determined by those parameters $\alpha_j$ whose corresponding features $f_j$ are active, that is $f_j(s,v) = 1$. We need to explore a set of features that can encode the information for some word types, called *word cluster*, that can help predict whether or not a word can occur in a nonzero state. For the first round of experiments, we used only ten features that they are most likely to be active on those phrase words. The generic feature function is defined as:

$$
f_j(s,v) = \begin{cases} 1 & \text{if } s \neq 0 \text{ and } (x(s) = 1 \text{ or } y(v) = 1) \\ 0 & \text{otherwise} \end{cases}
$$

where $x(s)$ represents a predicate function for the states and $y(v)$ is a predicate function for words. Table 5 lists the features.

| Feature | Function | Predicates |
|---|---|---|
| $f_1$ | $y_1(v)$ | if $v$ has a three-letter suffix used in noun |
| $f_2$ | $y_2(v)$ | if $v$ is capitalized |
| $f_3$ | $y_3(v)$ | if $v$ is all in uppercase |
| $f_4$ | $y_4(v)$ | if $v$ is an abbreviation |
| $f_5$ | $y_5(v)$ | if $v$ contains any digit |
| $f_6$ | $y_6(v)$ | if $v$ is hyphenated |
| $f_7$ | $y_7(v)$ | if $v$ has an apostrophe |
| $f_8$ | $x_1(s)$ | if $s = 1$ |
| $f_9$ | $x_2(s)$ | if $1 < s < 6$ |
| $f_{10}$ | $x_3(s)$ | if $s = 6$ |

Table 5: Feature functions

It's clear that $f_j(s,v)$ maybe active only if $s \neq 0$, or only when word $v$ is used by a phrase in the training data. So, after a number of iterations with GIS algorithm, the parameters $\alpha_{1,\ldots,10}$ are converged (see Section 5.3). The feature $f_{10}$ states that for all words that occur often in a phrase as head nouns (the last word), the terminal state is an important feature. To use these feature parameters as biases for word clusters, we modified the $b_{jk}$ (the observed probability of word $v_k$ being seen at state $s_j$) with an extra $\prod_{i=1}^{10} \alpha_i^{f_i(s_j, v_k)}$ when the Viterbi algorithm finds the best path for a word sequence. More details will give in the next subsection.

## 4.5 Scaling and Smoothing

As mentioned in (Rabiner, 1989), there is a potential problem in the probability calculations. Since all probabilities are less than 1.0, the numbers significantly less than one are being multiplied together, the values

will approach zero at an exponential rate, quickly exceeding the precision of machines. Hence a technique for scaling the Markov parameters $(A, B, \pi)$ is required to avoid mathematical underflow. A logarithmic computation is used to scale the probabilities so that at each time step of the Viterbi procedure it is a simple matter of adding the logarithms. For consistency, the biases of the ME parameters are also scaled by the logarithmic computation, thus, $\sum_{i=1}^{10} f_i(s_j, v_k) log(\alpha_i)$, and the sum will be added to the $b_{jk}$. This may not be necessary, for if the form (1) is not required in a probability mode then the normalization factor $1/Z(V)$ can be omitted from (1) and most of parameters $\alpha_{1,...,10}$ will not be very small (see Section 5.3).

Another problem in such a probabilistic model is that some of the probabilities are hard to estimate by direct counting, whatever the training data, because of Zipf's Law (frequency is roughly proportional to inverse rank) (Zipf, 1949). For example, considering a lexical probability, *(word frequency)/(total word frequency)*, because of Zipf's Law, no matter how much text we look at, there will always be a large tail of words that occur only few times, or will not occur at all (i.e. unknown words). Fortunately, several smoothing methods can help alleviate this problem. We used a simple way to smooth those unknown words and empty elements (i.e. $b_{jk} = 0$). The empty element $b_{jk} = 0$ means that a word $v_k$ has never been seen as $j$th word of a phrase during the training. Some empty elements are caused by the insufficient training data while some are caused by the words themself. For instance, the word *rivet* occurs in the training collection (see Section 4.2) four times but is not used as part of a noun phrase. In general text, however, there are many noun phrases containing this word, for example, *rivet body* and *blind rivet*, we call such a word non-phrase word. Some words can not be lexically used as $j$th word of a phrase. For example, adjectives can not be used as a head noun, and the preposition *of* can not be used at the beginning or ending of a phrase. This means that the smoothing algorithm should treat the empty elements differently.

Five "bins" or values based on the lexical occurrence probabilities are used for empty elements. The occurrence probability is computed during model initialization. The average probability, i.e. $avg_p = \sum_{v \in V} \tilde{p}(v)/z$ (recall that $z$ is the size of $V$, the training vocabulary), is used as a scaling factor, and the words are ranked from high occurrence probability to low. Thus five bins are:

| *Bin* | *Assumption* | *Range* |
|---|---|---|
| 1 | (well trained) | $(4avg_p, \infty)$ |
| 2 | (sufficiently trained) | $(2avg_p, 4avg_p]$ |
| 3 | (average trained) | $(avg_p/2, 2avg_p]$ |
| 4 | (insufficiently trained) | $(avg_p/4, avg_p/2]$ |
| 5 | (poorly trained) | $[1, avg_p/4]$ |

The empty elements (i.e. $b_{jk} = 0$) in the first bin (1) are not smoothed. The second bin is smoothed with the minimum probability, i.e. $min_p = \min_{v \in V} \tilde{p}(v)$. The empty elements in the third bin are smoothed with a word average probability, i.e. $avg_{v_k} = \sum_{i=0}^{6} b_{ik}/6$. Because of their importance to start and end phrases, the elements $b_{1k}$ and $b_{6k}$ for the second and third bins are not smoothed. For the fourth bin, the smoothing value used is also the word average probability, $avg_{v_k}$, but all empty elements are smoothed. For the fifth bin, a smooth value of $avg_p/4$ is added to every element of $b_{jk}$. There are an additional seven elements $b_{0,z+1}, ..., b_{6,z+1}$ for all unknown words (i.e. the words do not occur in the training data). These are all smoothed using the minimum probability $min_p$.

After smoothing the empty elements, the procedure normalizes every element $b_{jk}$ ($j = 0, 1, ..., 6; k = 1, ..., z, z+1$, plus the unknown word) using the total mass of probabilities of the words seen at the same state. The normalized $b'_{jk}$ are:

$$b'_{jk} = \frac{b_{jk}}{\sum_{k=1}^{z+1} b_{jk}}$$

The smoothing for the Markov parameter $B$ is accomplished by this normalization and the training process

of the Markov model is complete. With such a trained Markov model the Viterbi program can process the input tokens to output extracted phrases.

# 5 Experiments and Results

As mentioned in the Section 4, the TREC collections were used to generate a phrase dictionary. The 1987 Wall Street Journal collection was used for training the model. In addition to the training data, two test files were made for running experiments. The first is part of 1987 Wall Street Journal collection, called *"wsj-test"*, and includes 50 documents, 709 tokens, and 14,524 words, from which 654 longest noun phrases have been extracted manually. The other test file, called *"patent-test"* includes 21 text sections from Patent documents, 804 tokens, 2,049 words, and 733 longest noun phrases. The patent-test also includes 24 unknown words (i.e. never seen in the training), and 74 phrases contain these unknown words, 41 non-phrase words (i.e. never used by any phrase in the training), and 191 phrases containing them. The patent-test is considered a blind test since it was not used during the development of the system. A target file of noun phrases were extracted manually from each test file. The evaluation procedure is to compare the output of the phrase extractor with the target file.

Since the longest noun phrase in the target file is not specifically the goal, there is the problem of determining which extracted noun phrases can be regarded as *"correct"*. There are three simple standards for comparing the phrases in the output files with the ones in the target files.

1. **if** an extracted noun phrase matches the target one exactly **then** it is *correct*.
2. **if** an extracted noun phrase matches the last $n$ words ($n > 1$) of the target one **then** it is *correct*; for the proper names the output must match the target one completely.
3. **if** two sequentially output phrases match (using 1 or 2) both the words preceding and following a preposition *of* in a target phrase **then** they are counted as one *correct* match.

The output file is evaluated using recall and precision. Recall is the percentage of noun phrases in the target file that were correctly found by the phrase extractor. Precision is the percentage of the noun phrases found by the extractor that occurred in the target file. These measures provide a reasonable indication of performance, and the evaluation can be done automatically.

## 5.1 The Phrase Dictionary

Since the training dictionary plays a major role in our approach, the quality of the dictionary is very important. The dictionary is required not only a high "recall" but also a high "precision". That is, the dictionary is expected to contain as many correct noun phrase as possible. Using the heuristic method that was used for generating training dictionary, we collected noun phrases from the wsj-test file. The resulting phrase list had a recall of 97%, but the precision was only about 88%. When the *cutoff*=1, the recall was 21%, and the precision was 99%, which represented a significant tradeoff. Since the wsj-test has only at most 709 entries, it is possible to evaluate the output manually. It is more difficult, however, to choose a reasonable cutoff for 4,114,697 entries of the original training dictionary because it is too expensive to evaluate such a big dictionary manually. Using the Zipf's Law, we can remove the long tail in the frequency distribution. In the original dictionary there are 1,808,909 entries with frequency one, 469,036 entries with frequency two, and 216,237 entries with frequency three. The sum of these is 2,299,582 entries, which is almost 56% of the initial dictionary, so the cutoff value chosen was 3. The final dictionary after applying the cutoff has 1,815,115 entries.

Many other experiments have been done to improve the dictionary quality. For examples, using a POS tagger, Jtag (Xu, Broglio, & Croft, 1994) to tag tokens and remove those that are not a noun phrase pattern; using Mutual Information to predict a possible phrase; and using WordNet (Miller, Beckwith, Fellbaum, Gross, & Miller, 1990) to collect POS tags of every word and invoking a set of disambiguation rules to remove the verbs from tokens before doing the clustering. Compared with wsj-test results, using WordNet produced the best result (recall 100%, precision 97%), but this technique was not used to generate the training dictionary because of a speed concern[1].

All above experiments were done to generate a phrase dictionary in a four-step heuristic model (see Section 4.1). That is, hashing the tokens, removing non-noun ending words, clustering, and discarding the low frequency tokens. The different methods used in the experiments are applied only in step 2 to remove non-noun ending words. In fact, these methods can be applied to extract noun phrases directly. For instance, in previous work, there have been many studies using a POS tagger for the noun phrase extraction. The next section reports a comparison with this approach.

## 5.2 Comparison with POS Approaches

As part of this study, we developed a noun phrase extractor with Jtag. Jtag is a part-of-speech tagger developed by the Center for Intelligent Information Retrieval (CIIR) at University of Massachusetts at Amherst. Jtag uses a statistical ngram model similar to Church's work (Church, 1988). The major difference is that Jtag uses a bigram model instead of the trigram model used by Church.

The noun phrase extractor with Jtag performs well on the wsj-test and has recall 91% and precision 90%, while our heuristic model has recall 97% and precision 88%. From this experiment, however, we found three major problems affecting the performance of POS extractor (not only our Jtag extractor, which is just a instance here). These are sentence segmentation, lexicon ambiguity, and noun phrase patterns. Incorrect segmentation of sentences caused the tagger to make a number of mistakes. For example, consider the following text:

> *The contract consolidates 26 existing contracts that Boeing Computer Services has with NASA. Major subcontractors are Unisys, based in Detroit, and New Technology Inc., Huntsville, Ala., Boeing said.*

Looking at the capitalized word "Major" starting the second sentence, because of the segmentation error Jtag tags it as NP (annotation for proper noun), as follows

> *...has*/HVZ *with*/IN *NASA.*/NP *Major*/NP *subcontractors*/NNS *are*/BER...

Given this tagging, the Jtag extractor considers "*NASA. Major*" as a noun phrase. Correctly segmenting a sentence is difficult if the sentences end with an abbreviation of a proper name, or a short name, such as "*U.S.*", and "*Uncle Sam.*", because such proper names are very likely used to modify other names or nouns as well as to terminate a sentence. In these cases, since the sentence segmentation fail, the POS tagger is very difficult to assign correct tags to the capitalized words.

The second problem we found is that the statistical bigram model cannot disambiguate verb-noun lexicons very well because of limited training data and the ngram size. For example, consider the following tagged sentence

[1] Generating the dictionary from 131.7MB data of 1987 Wall Street Journal collection takes about 24 hours because the WordNet dictionaries are located on a hard disk and collecting POS tags for 6,125,944 words needs $4 \times 6,125,944$ binary searches on the disk

*The*/AT *company*/NN *designs*/NNS ,/, *manufactures*/NNS *and*/CC *markets*/NNS *pumps*/NNS ,/, *valves*/NNS *and*/CC *compressors*/NNS.

all three verbs *designs*, *manufactures*, and *markets* are tagged as NNS (plural noun) by Jtag. Jtag also makes such mistakes for many other words that can be used as either a verb or a noun, like *plan*, *move*, and *show*. Most of usage of these words are nouns in the training data, and the bigram cannot predict that they are used as verbs in this context because (NN NNS), (, NNS) and (CC NNS) are high frequent patterns. With these tagging mistakes, the extracted noun phrases are, of course, incorrect, for example, "*markets pumps*" and "*shows signs*".

Some of such mistakes can be corrected by the extractor with noun phrase patterns, for example, if there is no pattern for (NNS NNS) then "*markets pumps*" will not be extracted. However, some mistakes can not be corrected easily. For example, in the tagged clause,

*In*/IN *New*/NP *York*/NP *Stock*/NP *Exchange*/NP *composite*/NN *trading*/VBG ...

The extractor should contain *trading* in the noun phrase and output *New York Stock Exchange composite trading*. To do this, we cannot simply add a pattern (.. NN VGB), because this pattern will introduce more mistakes when a noun followed by a real gerund which leads a verb clause to modify the noun, such as

*department*/NN *managers*/NN *going*/VBG *up*/IN *the*/AT *ladder*/NN ...

The extractor should get only "*department managers*" rather than "*department managers going*". There are no complete definitions of noun phrase patterns. Even if the POS tagger was able to supply a 100% correctly tagged text there would still be problem in finding the right noun phrase patterns without syntax analysis.

Comparing POS approaches with our heuristic approach, all above three problems are difficult to solve, but the heuristic model used to generate the training phrase dictionary can overcome or avoid these difficulties. The heuristic model also requires the sentence segmentation but it does not make mistakes because of segmentation errors. This model identifies phrases based on the co-occurrence frequency of the words, so unless "*NASA. Major*" occurs enough times in the text, then the heuristic model could not consider it a phrase. The other two problems do not exist at all for the heuristic model, since it does not use syntactic knowledge. The heuristic model does, however, have limits. It is applicable only for large collections which contain enough co-occurrence information, and requires two passes of the collection to extract phrases. Thus the heuristic model cannot be used to extract phrases "on the fly". For example if an IR system wants to index phrases for a collection, it has to make a phrase dictionary on the first pass of the entire collection, and then create the index with dictionary lookups on the second pass. The heuristic model can also output incorrect noun phrases ending with non-noun words if they occur enough times in a collection. For example, "*company designs*" occurs 65 times in the TREC collections, if the cutoff value is less than 65 then it will be considered as a noun phrase. The higher cutoff value can improve the precision of extracted phrases but reduce the recall, such a tradeoff is another limit of the heuristic model. The heuristic model cannot extract phrases with both high recall and precision.

## 5.3 Markov Model Results

In this section we show some experiment results on the Markov model. There are two sets of experiment runs, one set for smoothing and the other one for biasing the model parameters. We did the smoothing experiments before the biasing experiments. However, to avoid the interaction between smoothing and biasing, the smoothing results shown in Table 6 come from the runs in which the ME parameter biases has been applied, and the biasing results shown in Table 7 come from the runs in which the five bin smoothing has been done.

### 5.3.1 Smoothing

The smoothing procedure is necessary for empty elements and unknown words as described in Section 4.5. The initial smoothing procedure simply filled every empty element $b_{jk}$ with its word average probability (i.e. $avg_{v_k}$). This caused many problems. For example, the word *"of"* in the training collection, it occurs 99,560 times at state two, 9,548 times at state three, 2,216 times at state four, and 218 times at state five, so the word average is very high. However, both $b_{1k}$ and $b_{6k}$ should be zero because *"of"* does not occur at the start or the end of a noun phrase. If the word average probability is used to fill these two empty elements, then incorrect phrases will be extracted. The Table 6 shows that five bin smoothing described in Section 4.5 performs better than this simple method.

| smoothing method | test | Recall% | Precision% |
|---|---|---|---|
| simple | wsj-test | 100 | 91 |
| five bin | wsj-test | 100 | 95 |
| simple | patent-test | 89 | 86 |
| five bin | patent-test | 93 | 91 |

Table 6: Evaluations of two smoothing methods

### 5.3.2 ME parameters

Before the ME model had been explored, we ran many experiments to figure out biases for the Markov parameters $A$ and $B$. Without the biases, the initialized Markov model extracts fewer phrases and gets low recall (68% on wsj-test, 34% on patent-test). The biases can increase parameters $A$ and $B$, and improve the recall and the precision, but they are neither robust nor constant. Table 7 shows that biases of 2, and 6 work very well on wsj-test but not nearly as well on patent-test, while biases of 9 and 9 perform reasonably well on patent-test but not on wsj-test.

| test | bias $A$ | bias $B$ | Recall% | Precision% |
|---|---|---|---|---|
| wsj-test (no bias) | 1 | 1 | 68 | 87 |
| patent-test (no bias) | 1 | 1 | 34 | 84 |
| wsj-test | 2 | 6 | 93 | 94 |
| patent-test | 2 | 6 | 78 | 89 |
| wsj-test | 9 | 9 | 99 | 88 |
| patent-test | 9 | 9 | 95 | 84 |

Table 7: Evaluations of two set of biases

We could not find a consistent set of biases that worked well for both of test files. For a more reliable and more flexible noun phrase extractor, the ME model was developed.

To train ME parameters, we ran an experiment with 400 iterations. Figure 2 shows that some of ME parameters are converged before 100 iterations while the others do not. It should also be noted that the parameters are not normalized, so that some of values are greater than 1.

Applying the ME parameters to bias the parameter $B$ of the Markov model can achieve good improvements on both test files. Specially, recall 100% and precision 95% on wsj-test, and recall 93% and precision 91% on patent-test. The patent-test result was the focus of the extractor evaluation because wsj-test is part of training data. For a summary, Table 8 compares the Markov model results with the other phrase extraction approaches for the paten-test file.

In Table 8, $correct_u/found_u$ is the ratio of number of correct phrases containing unknown words to the number of extracted phrases containing the unknown words, and $correct_p/found_p$ is the ratio for the non-

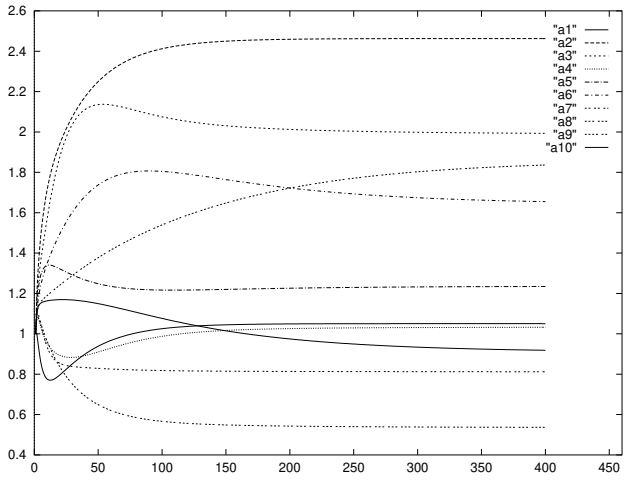


Figure 2: The parameter converging in 400 iterations

phrase words. In patent-test, there are 74 phrases containing unknown words and 191 phrases that contain non-phrase words. Overall, the comparison shows that the Markov model extractor performs better than the heuristic model, and very similar to the POS approach with WordNet. Almost all mistakes made by the Markov model extractor are caused by the unknown words and non-phrase words, even though the smoothing and optimizing procedures worked well. The difference between the results of the wsj-test and the patent-test indicates that the model needs optimization. The ME features offset insufficient training information.

| phrase extraction method | Recall% | Precision% | $correct_u/found_u$ | $correct_p/found_p$ |
|---|---|---|---|---|
| POS model with Jtag | 91 | 87 | 50/60 | 120/129 |
| heuristic model | 93 | 85 | 55/68 | 153/176 |
| MI model | 91 | 85 | 54/68 | 152/176 |
| POS model with WordNet | 90 | 94 | 51/56 | 130/132 |
| Markov model | 91 | 93 | 45/54 | 163/184 |

Table 8: Evaluations of four approaches for patent-test

# 6 Conclusion and Future Work

The Markov model approach provides an effective and trainable technique for noun phrase extraction. The extractor runs in a reasonable speed, is comparable in the effectiveness to the best results of other approaches, and is able to extract phrases on the fly. For an IR system, the extractor can index phrases without additional passes over the data.

The Markov model approach does not require syntactic knowledge or noun phrase patterns, unlike POS approaches. The most important factors for the Markov model are the size and quality of the training dictionary. Analysis of errors made by the current version of the extractor confirms that the quality of the training dictionary should be improved in order to further improve performance. One way of avoiding the problem of dictionary quality is to train the Markov model without the dictionary, that is unsupervised training. The trained model would be a Hidden Markov model, which could be optimized by using forward-backward Baum-Welch algorithm. This will be studied in future work.

The ME model is an extremely flexible technique for linguistic modeling, since it can use a virtually unrestricted and rich feature set in the framework of a probability model. Our current feature set is not rich enough yet. Another area for future research is to discover more lexicon features or word-based features for identifying the

word clusters for the optimization with the Maximum Entropy model. For example, we could use every suffix instead of noun-suffix only, and add more contextual features, etc.

There may also be a more flexible smoothing method than the five bin approach. Moreover, experiments involving smoothing for the transition parameter $A$ need to be done.

# References


Abney, S. (1990). Rapid increamental parsing with repair. In *Proceeding, Waterloo Conference on Electronic Text Research.*

Baum, L. E. & J. Eagon (1963). An inequality with applications to statistical prediction for functions of markov processes and to a model for ecology. *Bulletin American Mathematic Society 73*, 360–363.

Baum, L. E. & J. Eagon (1972). An inequality with associated maximization technique in statistical estimation for prediction functions of markov processes. *Inequalities 3*, 1–8.

Bourigault, D. (1992). Surface grammatical analysis for extraction of terminological noun phrases. In *Proceedings of the Fourteenth COLING*, pp. 977–981.

Brants, T. (2000). TnT - a statistical part-of-speech tagger. In *http://www.coli.uni-sb.de/ thorsten/publications/Brants-ANLP00.pdf.* Saaland University.

Brill, E. (1993). *A corpus-based approach to language learning.* Ph. D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Charniak, E., C. Hendrickson, N. Jacobson, & M. Perkowitz (1993). Equations for part of speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, pp. 784–789.

Choueka, Y. (1988). Looking for needles in a haystack. In *Proceeding, RIAO Conference on User-Oriented Context Based Text and Image Handling*, Cambridge, MA, pp. 609–623.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted test. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pp. 136–143. Association for Computational Linguistics.

Cutting, D., J. Kupiec, J. Pedersen, & P. Sibun (1992). A pratical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, Trento, Italy. Association for Computational Linguistics.

Darroch, J. & D. Ratcliff (1972). Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics 43*(5), 1470–1480.

DeRose, S. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics 14*, 31–39.

Fagan, J. (1989). The effectiveness of a nonsyntactic approach to automatic phrase indexing for document retrieval. *Journal of the American Society for Information Science 40*(2), 115–132.

Francis, W. & F. Kučera (1982). *Frequency Analysis of English Usage: Lexicon and Grammar.* Boston, MA: Houghton Mifflin.

Haase, K. B. (1996). Framerd: Representing knowledge in the large. *IBM Systems Journal 35(3/4)*, 381–397.

Harman, D. (1995). Overview of the second text retrieval conference (trec-2). *Information Processing and Management 31*(3), 271–289.

Jaynes, E. T. (1957a). Information theory and statistical mechanics: Part I. *Physical Review 106*, 620–630.

Jaynes, E. T. (1957b). Information theory and statistical mechanics: Part II. *Physical Review 108*, 171.

Jelinek, F. (1985). Markov source modeling of text generation. *Impact of Proceeding Techniques on Communication E91 of NATO ASI series*, 569–598.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden markov model. *Computer, Speech, and Language 6*, 225–242.

Kupiec, J. (1993). Murax: A robut linguistic approach for question answer using an on-line encyclopedia. In R. Korfhage, E. Rasmussen, & P. Willett (Eds.), *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Pittsburgh, Pa. USA, pp. 160–169.

Lau, R., R. Rosenfeld, & S. Roukos (1993). Adaptive language modeling using the maximum entropy principle. In *Proceeding of the Human Language Technology Workshop*, pp. 108–113. Advanced Research Projects Agency.

Merialdo, B. (1994). Tagging english text with a probability model. *Computational Linguistics 20*(2), 155–172.

Miller, G., R. Beckwith, C. Fellbaum, D. Gross, & K. Miller (1990). Five papers on the wordnet. Technical Report July, Princeton University, Computer Science Laboratory.

Perez-Carballo, J. & T. Strzalkowski (2000). Natural language information retrieval: progress report. *Information Processing and Management 36*(1), 155–178.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceeding of the IEEE 77*(2), 257–286.

Rabiner, L. & B. Juang (1986). An introduction to hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing January*, 4–16.

Ratnaparkhi, A. (1998). Statistical models for unsupervised prepositional phrase attachement. In *ACL 36/COLING 17*, pp. 1079–1085. Association for Computational Linguistics.

Ratnaparkhi, A., J. Reynar, & S. Roukos (1994). A maximum entropy model for prepositional phrase attachment. In *Proceeding of the Human Language Technology Workshop*, Plainsboro, NJ, pp. 250–255. Advanced Research Projects Agency.

Salton, G. & M. Lesk (1968). Computer evaluation of indexing and text processing. *Association for Computing Machinery 15*, 8–36.

Smadja, F. (1993). Retrieving collocations from text: Xtract. *Computational Linguistics 19*(1), 143–177.

Tolle, K. M. & H. Chen (2000). Comparing noun phrasing techniques for use with medical digital library tools. *Information Processing and Management 51*(4), 352–370.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory IT-13*, 260–269.

Voutilainen, A. (1997). A short introduction to nptool. In *http://www.lingsoft.fi/doc/nptool/intro*.

Weischedel, R., M. Meteer, R. Schwartz, L. Ramshaw, & J. Palmucci (1993). Coping with ambiguity and unknown words through probability model. *Computational Linguistics 19*(2), 359–382.

Xu, J., J. Broglio, & W. Croft (1994). The design and implementation of a part of speech tagger for english. Technical Report IR-52, University of Massachusetts, CIIR.

Zipf, G. (1949). *Human behavior and the principle of least effort*. addison-wesley.