ESTIMATED RANK PRUNING AND JAVA-BASED SPEECH RECOGNITION

Nikola Jevtić   Aldebaro Klautau   Alon Orlitsky

ECE Department, UCSD
9500 Gilman Drive
La Jolla, CA 92093, USA

ABSTRACT

Most speech recognition systems search through large finite state machines to find the most likely path, or hypothesis. Efficient search in these large spaces requires pruning of some hypotheses. Popular pruning techniques include probability pruning which keeps only hypotheses whose probability falls within a prescribed factor from the most likely one, and rank pruning which keeps only a prescribed number of the most probable hypotheses. Rank pruning provides better control over memory use and search complexity, but it requires sorting of the hypotheses, a time consuming task that may slow the recognition process. We propose a pruning technique which combines the advantages of probability and rank pruning. Its time complexity is similar to that of probability pruning and its search-space size, memory consumption, and recognition accuracy are comparable to those of rank pruning. We also describe a research-motivated Java-based speech recognition system that is being built at UCSD.

1. INTRODUCTION

Speech recognition systems commonly model speech as *finite state machines (FSM's)*. The recognition (or decoding) uses a *Viterbi search* to find the most likely path, or *hypothesis* in the FSM. However, in many applications the FSM's are exceedingly large and exhaustive search is not feasible. Sub optimal search algorithms are used instead, e.g., [1, 2].

The most common are the *probability pruning*, or *Viterbi beam search*, algorithms. They are given a prescribed *probability threshold* T, and, at each step, they extend all previous hypotheses and maintain only the T*-probable* ones—those whose log probability is at most T lower than that of the most likely hypothesis. By contrast, *rank, or histogram pruning* algorithms are given a desired number $N_{\rm opt}$ of hypotheses, and at each step, they extend all previous hypotheses and maintain only the $N_{\rm opt}$ most likely ones. Rank pruning is usually combined with probability pruning so that only the T-probable among the $N_{\rm opt}$ most likely hypotheses are maintained.

Probability pruning is fast, but may maintain an unlimited number of hypotheses, which can take longer to search and require more storage. Rank pruning maintains a fixed number ($N_{\rm opt}$) of hypotheses, hence better controls memory usage and the size of the search space, but, at every frame, it uses sorting to determine the most probable hypotheses. Sorting is time consuming and may considerably slow recognition.

We propose an *estimated rank* pruning technique which combines the advantages of probability and rank pruning. Its time complexity is similar to that of probability pruning and its search-space size, memory consumption, and recognition accuracy are comparable to those of rank pruning. For example, in experiments we performed, estimated rank pruning used an average of 6% more memory than rank pruning, the set of hypotheses it maintained differed by at most 7% from those maintained by rank pruning, its recognition accuracy was essentially the same, and its running time was similar to that of probability pruning and about 50% lower than that of rank pruning.

We implemented and tested the estimated rank pruning on a speech recognition system that is being built at UCSD. The system is Java based and uses an intuitive graphical user interface to facilitate experimentation with a variety of speech recognition architectures and parameters.

The estimated rank pruning technique is described next. In Section 3 we describe the results obtained and compare them to rank and hypothesis pruning. In Section 4 we briefly describe the system used to implement the algorithm and in Section 5 we mention the actual parameters it used.

2. ESTIMATED RANK PRUNING

Recall that an hypothesis is t*-probable* if its log probability is at most t below that of the most probable hypothesis at its step. Let $N_i(t)$ be the number of t-probable hypotheses at step i and let t_i be the difference between the log probability of the most likely hypothesis at time i and that of the $N_{\rm opt}$'th most likely one at that step. Note that $N_i(t_i) = N_{\rm opt}$.

Had we known t_i, we could efficiently mimic rank pruning by keeping only the the t_i-probable hypotheses. But t_i is not known, and exact calculation of t_i by sorting would use the very operation we are trying to avoid. Instead, estimated rank pruning uses a fast calculation to find a threshold $\hat{t}_i$ such that $N_i(\hat{t}_i) \approx N_{\rm opt}$.

To quickly determine $\hat{t}_i$, estimated rank pruning uses an empirical observation indicating that $N(t)$ is roughly exponential in t. Figure 1 shows a semi-log scale plot of $N_i(t)$ as a function of t for several frames of a typical sentence. This relation appears in almost all sentences and frames. It suggests that over short threshold intervals, $\log N_i(t)$ grows roughly linearly with t, namely,

$$N_i(t) \approx a_i \cdot e^{b_i \cdot t}, \tag{1}$$

for some constants a_i and b_i. It follows that if we take two thresholds t' and t'' near (the unknown) t_i, we can evaluate

$$b_i = \frac{\log N_i(t') - \log N_i(t'')}{t' - t''} \quad \text{and} \quad a_i = N_i(t') \cdot e^{-b_i t'}. \tag{2}$$

We then use Equation (1) to determine

$$\hat{t}_i = \frac{1}{b_i} \ln \frac{N_{\rm opt}}{a_i}. \tag{3}$$

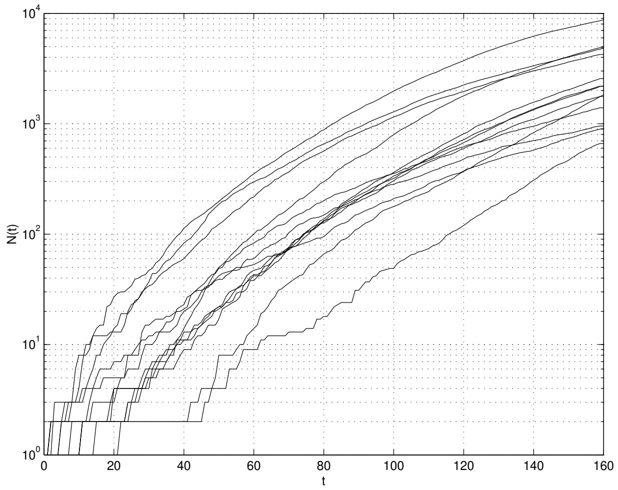

Fig. 1. Number of hypotheses as a function of the threshold

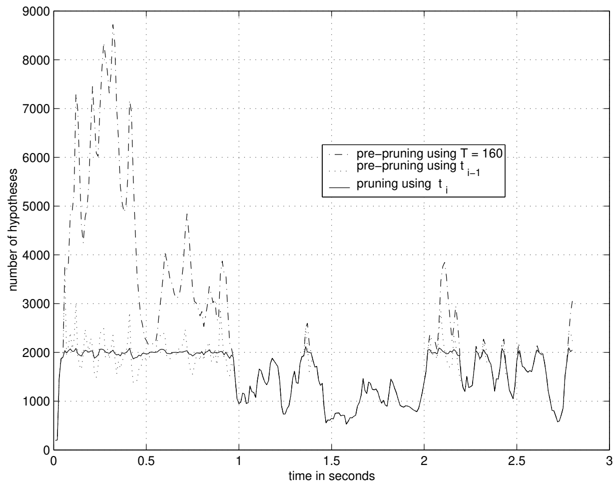

Fig. 2. Hypotheses selected by different thresholds ($N_{\rm opt} = 2000$)

We then use $\hat{t}_i$ as our estimate of t_i and maintain only the $\hat{t}_i$-probable hypotheses.

In our implementation we choose $t' = t_{i-1}$ and $t'' = (1-\delta) \cdot t_{i-1}$ where δ is a small constant, usually 0.2. Typically, t_i is not far from t_{i-1}, hence t' and t'' are fairly close to t_i. Note that $\hat{t}_{i-1}$ is not used to estimate $\hat{t}_i$, only to determine the points t' and t'' on which the estimate of $\hat{t}_i$ is based. Hence, if $\hat{t}_i$ is very different from $\hat{t}_{i-1}$, the estimate will be less reliable, but not necessarily biased.

We investigated two versions of the algorithm. Both increase efficiency by pre-pruning while extending the hypotheses. The *conservative version* (*CV*) keeps all T-probable hypotheses, while the *efficient version* (*EV*) keeps only the $\hat{t}_{i-1}$-probable hypotheses.

The protocols can be summarized as follows:

At frame i:

1. Starting with the most likely and continuing in any order, extend all hypotheses keeping only those within
   CV: T
   EV: $\hat{t}_{i-1}$
   from the best extended hypothesis thus far.
2. Set $t' = \hat{t}_{i-1}$ and $t'' = (1-\delta) \cdot \hat{t}_{i-1}$.
3. Calculate $N_i(t')$ and $N_i(t'')$.
4. Determine a_i and b_i using Equation (2).
5. Use Equation (3) to determine $\hat{t}_i$.
6. Set $\hat{t}_i = \min(\hat{t}_i, T)$.
7. Prune all but the $\hat{t}_i$-probable hypotheses.

Both algorithms run efficiently. Step 1 is the same as in other Viterbi searches. Steps 2, 4, 5, and 6 take a small constant time. Steps 3 counts hypotheses and is linear in their number. Step 7 eliminates all non t-probable hypotheses hence takes linear time as well; it is common to all search algorithms, and in practice is combined with step 1.

Step 1 describes the pre-pruning. Note that algorithm CV removes only hypotheses that are not T-probable. It follows that all $\hat{t}_i$-probable hypotheses are preserved, hence assuming that $\hat{t}_i$ is close to the actual threshold, we will end the iteration with roughly $N_{\rm opt}$ hypotheses.

While CV is faster than rank pruning, it still requires as much memory as probability pruning. EV addresses this by using a stronger pre-pruning step. It assumes that the previous threshold is close to the current one, and prunes all hypotheses that are not $\hat{t}_{i-1}$-probable. However when $\hat{t}_{i-1} < \hat{t}_i$ it may pre-prune some $\hat{t}_i$-probable hypotheses which should have been kept, namely those that are not $\hat{t}_{i-1}$-probable[1].

Figure 2 shows the effects of pre-pruning in the two versions of the algorithm. The dashed, solid, and dotted lines show the number of hypotheses maintained by CV after pre-pruning, by CV at the end of each iteration, and by EV after pre-pruning respectively. Whenever the dotted line falls below the solid line, the pre-pruning step of EV eliminates hypotheses that should have been preserved.

CV and EV can be viewed as the two extremes of optimizing the accuracy of hypotheses pruning (CV) and minimizing memory use (EV). Intermediate algorithms can also be considered.

The next section described the results obtained using estimated rank pruning. Section 4 describes the system used to run the tests and Section 5 describes the actual parameters used.

3. SIMULATIONS AND RESULTS

We tested the algorithms on the TIMIT database (for a precise description see Section 5).

We compared the performance of rank pruning to that of the two estimated rank pruning algorithms. We targeted the same number of maintained hypotheses in all algorithms. The results show that the WER is roughly the same across the three algorithms, but run times and memory consumption varied significantly. Algorithm CV runs faster than rank pruning, but uses more memory. EV runs even faster than CV and on the average uses only marginally more memory than rank pruning.

Figure 3 plots the run times of the three algorithms relative to the actual duration of the recognized speech. The global threshold T was fixed at 160, and $N_{\rm opt}$ varied from 500 to 5000. As can be

[1] Some of these hypotheses may still survive as the probability of the best hypothesis extended before them may be lower than that of the best extended hypothesis at the end.

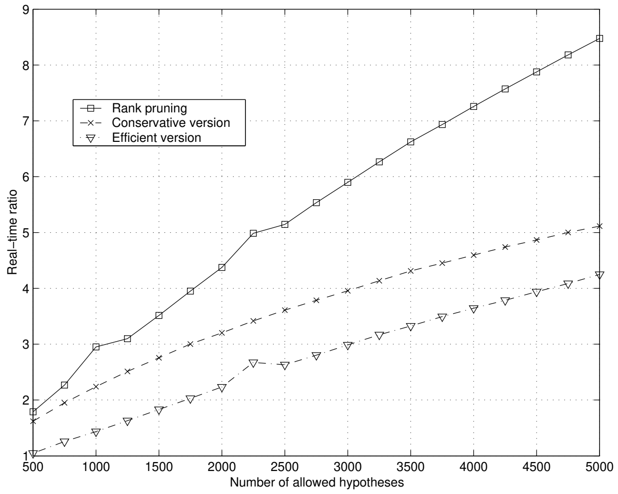

Fig. 3. Time performance of the three decoders

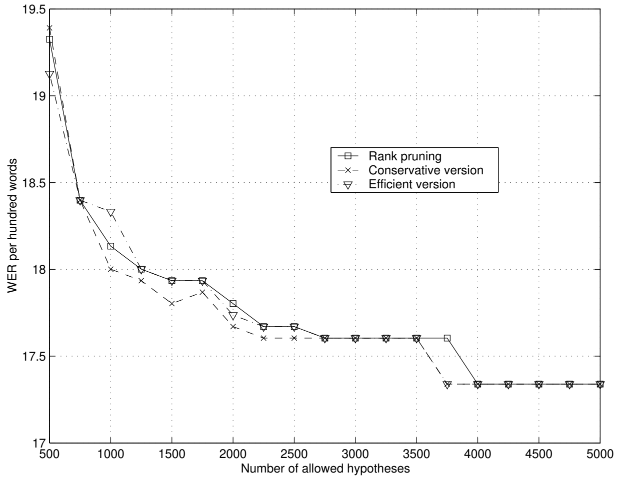

Fig. 4. Recognition performance of the three decoders

observed, EV is consistently about twice as fast as rank pruning, and about one real time faster than CV. For example, with a global threshold $T = 160$ and $N_{\text{opt}} = 2000$, rank pruning runs at 4.37 times real time, CV runs at 3.2 times real time, EV at 2.23.

The difference between the run time of CV and EV can be explained by the more aggressive pre-pruning in EV. It results in fewer hypotheses that need to be counted, fewer hypotheses at the end of each step, and less work for Java's garbage collector.

Figure 4 plots the WER of the three algorithms for the same range. As can be observed, the three algorithms have similar accuracies. For example, with the same parameters as above, the WER's were 17.8% rank pruning, 17.67% CV and 17.74% EV.

Next we evaluate the memory usage of the two estimated pruning algorithms and their accuracy in finding the most likely hypotheses.

Let M_i denote the number of hypotheses maintained by the estimated pruning algorithms at frame i, just after Step 1, and define $miss = \frac{|N_i(\hat{t}_i) - N_{\text{opt}}|}{N_{\text{opt}}}$, and $over = \max\left(0, \frac{M_i - N_{\text{opt}}}{N_{\text{opt}}}\right)$. When computing their average values, $miss$ is averaged over all frames when $t_i < T$, namely where $N_i(T) > N_{\text{opt}}$, and $over$ is averaged over all frames since we may have $M_i \geq N_{\text{opt}}$ even when $N_i(T) < N_{\text{opt}}$ (due to the uncertainty of the best score during pre-pruning step).

Table 1 shows the average and peak number of hypotheses kept in the list in excess to N_{opt}. Average values of EV suggest that the method has successfully addressed the memory limitations. The fact that peak values are high is not alarming since the average values show that it can not be frequent. In designing the system that uses EV, we should still be prepared to handle more hypotheses then N_{opt}.

In Table 2 we show the absolute error, relative to N_{opt}, in rank estimation. CV shows that a good estimate is possible as the average absolute error is less then 2.5%. The increased average error in EV comes from the pre-pruning stage. As mentioned before, this can be brought as close to performance of CV at the expense of increasing memory.

This difference is reflected in the WER as shown in Figure 4. CV outperforms EV in almost every point. However, EV performs roughly the same as rank pruning. It is not clear if CV systematically outperforms rank pruning by coincidence. We have not observed significant increase in the average number of hypotheses extended by CV. There may therefore be an unobserved connection between the way we determine probability and the nature of recognition process that helps increasing the count only when it is beneficial for the recognizer.

If we are to estimate the effects of this technique applied to large vocabulary systems, we could say that in absolute numbers the savings would be higher. Mainly because N_{opt} is usually around 10000 in LVCSR, and sorting complexity increases faster then linearly. On the other hand, relative increase in overall system performance may not be so high since large vocabulary systems usually use more complex phone models (like triphones, pentaphones) and spend most of the recognition time in computing models.

4. THE SYSTEM

The algorithm was tested on a speech-recognition system that is being built at UCSD. The main purpose of the system is to facilitate speech recognition research. A state-of-art ASR system can easily exceed 50,000 lines of code. Additionally, in some sites, especially universities, the developers change frequently. These factors call for the construction of a well-structured and documented code, with a reasonable learning curve. Based on these factors, the Java platform was selected. Java is easy to learn, use, and maintain. It is portable, object-oriented, and supports networked applications and graphical user interfaces. Currently, Java is slower than C. In some benchmarks we ran with a *Mel Frequency Cepstral Coefficient* (*MFCC*) front end, Java was 1.5 to 3 times slower than C, but the gap has been shrinking as Java compilers have evolved.

The system utilizes the three traditional modeling stages: front end, acoustic, and language. Within each stage, the system lets the user select among several options.

The front-end module implements several feature sets. MFCC's with normalized energy and derivatives obtained through linear regression [3], PLP [4], RASTA [5], LSF [6], and Seneff's auditory model [7]. Some of these features are implemented in Java, while others use interfaces to publicly available software which the system also supports.

$N_{\rm opt}$	Conservative version		Efficient version	
	avg. over	max over	avg. over	max over
500	241.6%	1187.0%	9.4%	411.8%
750	177.3%	866.0%	8.6%	244.7%
1000	139.9%	721.9%	8.0%	226.7%
1250	115.2%	615.4%	7.6%	214.6%
1500	97.6%	546.5%	7.2%	203.5%
1750	84.3%	494.4%	6.9%	207.4%
2000	73.8%	458.9%	6.5%	197.8%
2250	65.4%	422.5%	6.2%	181.2%
2500	58.4%	396.0%	5.9%	185.7%
2750	52.6%	371.5%	5.6%	192.5%
3000	47.5%	356.4%	5.4%	180.0%
3250	43.2%	334.7%	5.1%	158.4%
3500	39.4%	320.4%	4.9%	139.9%
3750	36.0%	308.6%	4.6%	123.9%
4000	33.1%	293.8%	4.4%	110.0%
4250	30.4%	280.8%	4.2%	97.6%
4500	28.1%	271.8%	4.0%	94.0%
4750	26.0%	263.9%	3.8%	89.9%
5000	24.1%	252.3%	3.6%	82.4%

Table 1. Extra hypotheses stored

$N_{\rm opt}$	Conservative version		Efficient version	
	avg. miss	max miss	avg. miss	max miss
500	2.5%	97.0%	8.2%	97.6%
750	2.3%	73.7%	7.8%	75.6%
1000	2.2%	45.9%	7.7%	71.2%
1250	2.2%	27.0%	7.6%	69.5%
1500	2.2%	25.8%	7.5%	66.4%
1750	2.3%	26.3%	7.5%	66.5%
2000	2.3%	23.9%	7.4%	66.1%
2250	2.3%	24.8%	7.4%	66.5%
2500	2.4%	25.8%	7.3%	66.0%
2750	2.4%	25.0%	7.2%	65.2%
3000	2.4%	23.2%	7.1%	64.8%
3250	2.4%	24.1%	7.1%	65.2%
3500	2.4%	23.0%	7.0%	64.4%
3750	2.4%	23.4%	6.9%	63.7%
4000	2.4%	21.9%	6.8%	63.5%
4250	2.5%	21.8%	6.7%	63.1%
4500	2.5%	20.9%	6.7%	62.6%
4750	2.5%	20.1%	6.6%	62.4%
5000	2.5%	20.6%	6.6%	62.0%

Table 2. Absolute estimation error

The acoustic models support continuous *hidden Markov models* (*HMMs*) with and without state sharing. The HMMs can be re-estimated with the isolated or embedded Baum-Welch algorithm. The models can be initialized by Gaussian splitting or a segmental K-means procedure. Training can be conducted through a user-friendly GUI that also provides a summary of the results.

The language-model options include n-grams of arbitrary depth, Katz's back-off model, and several variations of Good-Turing probability estimates. Two options for *language model look-ahead* [8] are also available.

The decoder is frame based and contains options for probability pruning and exact and approximate rank pruning as explained.

5. PARAMETERS

To evaluate estimated rank pruning, we tested the two algorithms on the TIMIT database. We note however that the test was somewhat unfair as the recognizer used pronunciation and language models based partly on the test set. Therefore the results described reflect the relative merits of the various pruning techniques and should not be used for comparison with other baseline systems.

The front end used 39-dimensional feature vectors, 12 mel-based cepstral coefficients, energy, and their first and second derivatives. Frame width was 25ms and frame shift was 10ms. The system used 48 speaker-independent monophones as defined in [9]. They were modeled as 3-state, left-to-right, HMMs with at most ten Gaussian mixture components. Short pause was modeled as a single state HMM. The models were trained on the training portion of TIMIT. The tests were performed on TIMIT core test set.

The language model used bigrams based on both the training and testing sets. Its back-off probabilities were determined by a Good-Turing estimate. The total number of words in TIMIT is 6100, but due to the size of the data only about 950 contexts were created. The average perplexity of the core test set was 1741.

The pronunciations dictionary used all pronunciations appearing in both the training and testing sets according to the hand-labeled transcriptions. Each pronunciation was assigned a probability proportional to its frequency. The final network was obtained by composition of the language model network and the pronunciation dictionary. The total number of vertices in final network was 112138 and the number of edges was 184450. Hypotheses corresponded to edges.

The test was performed on a Pentium 3 500MHz computer with 128MB of memory.

6. REFERENCES

[1] H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16(5):64–83, Sept. 1999.

[2] N. Desmukh, A. Ganapathiraju, and J. Picone. Hierarchical search for large-vocabulary conversational speech recognition: working toward a solution to the decoding problem. *IEEE Signal Processing Magazine*, 16(5):84–107, Sept. 1999.

[3] J. Picone. Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9):1215–47, Sep. 1993.

[4] H. Hermansky. Perceptual linear predictive (plp) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738–52, Apr. 1990.

[5] H. Hermansky and N. Morgan. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing*, 2(4):578–89, Oct. 1994.

[6] F. Soong and B.-H. Juang. Line spectrum pair (lsp) and speech data compression. *ICASSP 84*, 1:p.1.10/1–4, Mar. 1984.

[7] S. Seneff. Pitch and spectral estimation of speech based on auditory synchrony model. *ICASSP 84*, 3:p.36.2/1–4, Mar. 1984.

[8] H. Ney and S. Ortmanns. Look-ahead techniques for fast beam search. *Computer speech and language*, 14(1):15–32, Jan. 2000.

[9] K.-F. Lee and H.-W. Hon. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11):1641–8, Nov. 1989.