

Speeding Up Dynamic Search Methods in Speech Recognition

Gábor Gosztolya and András Kocsor

MTA-SZTE Research Group on Artificial Intelligence,
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{ggabor, kocsor}@inf.u-szeged.hu

Abstract. In speech recognition huge hypothesis spaces are generated. To overcome this problem dynamic programming can be used. In this paper we examine ways of speeding up this search process even more using heuristic search methods, multi-pass search and aggregation operators. The tests showed that these techniques can be applied together, and their combination could significantly speed up the recognition process. The run-times we obtained were 22 times faster than the basic dynamic search method, and 8 times faster than the multi-stack decoding method.

In speech recognition enormous hypothesis spaces arise. To handle them we can use dynamic programming, where we can avoid calculating the same values several times, which leads to a dramatic speed-up of a speech recognizer system. But this is not enough for real-world applications, hence we have to look for other ways of making improvements while preserving the recognition accuracy. Here we carry out experiments using search heuristics, aggregation operators and multi-pass search, and apply ideas for speeding up the heuristic search.

1 The Speech Recognition Problem

We have a speech signal given by a series of observations $A = a_1 \dots a_t$, and a set of phoneme sequences W . We look for the word $\hat{w} \in W = \arg \max P(w|A)$ which, via Bayes' theorem, is equivalent to $\hat{w} = \arg \max (P(A|w) \cdot P(w))/P(A)$. $P(A)$ is the same for all w , so $\hat{w} = \arg \max P(A|w)P(w)$. Let w be $o_1 o_2 \dots o_n$, as o_j is the j th phoneme of w . Let A_1, \dots, A_n be non-overlapping segments of A . We assume that the phonemes are independent, i.e. $P(A|w)$ can be obtained from $P(A_1|o_1), \dots, P(A_n|o_n)$. To calculate $P(A|w)$, we can use aggregation operators at two levels: g_1 supplies the $P(A_j|o_j)$ values as $g_1(P(a_{t_{j-1}}|o_j), \dots, P(a_{t_j}|o_j))$, while g_2 is used to construct $P(A|w)$ as $g_2(P(A_1|o_1), \dots, P(A_n|o_n))$.

Instead of a probability p we will use a cost $c = -\ln p$. g_1 will be the addition operator. A *hypothesis* is a pair of phoneme series and segment series. The dynamic programming method uses a table with the a_i speech frames indexing the columns and the phoneme-sequences indexing the rows. A cell holds the lowest cost of the hypotheses having its phoneme-sequence and ending at its frame. To compute the value of a cell we take the value of an earlier frame and its

phoneme-sequence without its last phoneme, and add up the cost of this last phoneme on the interleaving frames. The result is the minimum of these sums.

2 Speeding Up the Recognition Process

The dynamic programming search technique, despite its effectiveness, tends to be quite slow. In this section we discuss some methods that speed it up while keeping the recognition accuracy at an acceptable rate.

Heuristic Search Methods. These techniques fill only a part of the table. So the result will not always be optimal, but we can get a notable speed-up with little or no loss in accuracy. The multi-stack decoding algorithm fills a fixed number (*stack size*) of cells (the ones with the lowest costs) for a row. The Viterbi beam search fills the cell with the best value, and the cells close to it defined by a *beam width* parameter. Here we used the multi-stack approach.

Speed-Up Improvements. In earlier works [1] we presented some speed-up ideas for the multi-stack decoding algorithm, which we also want to use here.

i) One possibility is to combine multi-stack decoding with a Viterbi beam search. At each column, belonging to one time instance, we fill only a fixed number of cells, and also discard those which are far from the best-scoring value.

ii) Another approach is based on the fact that the later the time instance, the fewer hypotheses (and filled cells) are need. Thus we filled $s \cdot m^i$ cells belonging to the a_i frame, where $0 < m < 1$ and s is the original *stack size* parameter.

iii) Actually, we need to fill more cells at those speech frames close to pronounced phoneme bounds. We trained an ANN to estimate whether a given time instance was a phoneme bound or not. Then we constructed a function that approximates the stack size based on the output of this ANN.

Multi-pass Search. Multi-pass methods work in several steps: in the first pass the worse hypotheses are discarded because of some condition requiring low computational time. We reduced the number of phoneme groups for this reason. In later passes only the remaining hypotheses are examined, but with a more detailed phoneme grouping. The last pass (\mathcal{P}_0) uses the original phoneme set. To create the phoneme-sets first a distance function of the original ph_1, \dots, ph_m phonemes is defined: $d(ph_i, ph_j)$ is based on the ratio of ph_i -s classified as ph_j and vice versa. We can use the higher value (d^1) or the average (d^2) as the metric. The distance between phoneme-groups can be the minimum distance between their phones (\mathcal{D}_{min}), or the maximum (\mathcal{D}_{max}) [2]. The recognition steps using the resulting phoneme-sets were \mathcal{P}_1 and \mathcal{P}_2 .

3 Tests and Results

The train database consisted of 500 speakers, each uttering 10 sentences via telephone. In the test database the 431 speakers uttered the name of a town.

Table 1. Recognition results. The basic dynamic search method resulted in 431,607.07 phoneme-identifications, while the Viterbi beam search produced 131,791.63

Phoneme group		Passes			Used Improvements			
		\mathcal{P}_0	\mathcal{P}_1	\mathcal{P}_2	–	i	iii	ii
standard		•	◦	◦	169,330.43	72,199.19	58,735.97	55,702.61
d^1	\mathcal{D}_{min}	•	•	◦	110,300.97	32,382.85	30,727.94	30,103.32
		•	◦	•	–	–	–	–
		•	•	•	–	–	–	–
	\mathcal{D}_{max}	•	•	◦	111,047.41	26,591.38	20,769.16	19,306.91
		•	◦	•	135,975.42	62,053.11	53,021.70	51,019.48
		•	•	•	170,505.40	70,249.03	61,114.36	59,737.51
d^2	\mathcal{D}_{min}	•	•	◦	111,042.23	26,920.40	20,857.46	19,327.69
		•	◦	•	–	–	–	–
		•	•	•	–	–	–	–
	\mathcal{D}_{max}	•	•	◦	91,889.07	47,328.51	38,515.23	36,914.01
		•	◦	•	217,525.55	98,423.11	78,825.82	76,961.10
		•	•	•	216,652.05	107,467.50	88,106.10	87,416.17

The *HTK* system [3] yielded 92.11% here. We first improved the recognition rate with aggregation operators [1], then the multi-stack decoding algorithm was used with the lowest stack size that kept the optimal accuracy. Next, multi-pass tests were applied. After we used the speed-ups in the sequence described in [1]. The speed of a configuration was the lowest one with accuracy above 92%, and was measured in average phoneme-identifications normalized to the last pass. We see that only those multi-pass configurations including \mathcal{P}_2 were unsuccessful. Using both the multi-stack decoding algorithm and the Viterbi beam search (improvement i) resulted in a 48-76% reduction in running times. Improvement iii reduced running times by 20%, and improvement ii also produced a slight speed-up.

4 Conclusion

In this paper we examined a dynamic search method, and some ways of speeding up this search process. We employed several tools like heuristic search, aggregation operators, multi-pass search and other ideas, which resulted in a dramatic speed-up with the same level of accuracy. In the end our method proved to be 22 times faster than the dynamic search algorithm, 6 times than the Viterbi beam search, and 8 times faster than the multi-stack decoding method.

References

1. G. GOSZTOLYA, A. KOCSOR, *Aggregation Operators and Hypothesis Space Reductions in Speech Recognition*, Proc. of TSD, LNAI 3206, pp. 315-322, Springer, 2004.
2. G. GOSZTOLYA, A. KOCSOR, *A Hierarchical Evaluation Methodology in Speech Recognition*, Submitted to Acta Cybernetica, 2004.
3. S. YOUNG ET AL., *The HMM Toolkit (HTK) (software and manual)*, <http://htk.eng.cam.ac.uk/>