Acta Cybernetica **16** (2003) 229-240.

Various Robust Search Methods in a Hungarian Speech Recognition System*

Gábor Gosztolya† András Kocsor† László Tóth†

and László Felföldi‡

Abstract

This work focuses on the search aspect of speech recognition. We describe some standard algorithms such as stack decoding, multi-stack decoding, the Viterbi beam search and an A* heuristic, then present improvements on these search methods. Finally we compare the performance of each algorithm, grading them according to their performance. We will show that our improvements can outperform the standard methods.

KeyWords. search methods, stack decoding, multi-stack decoding, Viterbi beam search.

1 Introduction

In any speech recognition system, the real task is to find the most probable word (sequence of phonemes) for a given speech signal. However, as the number of possibilities is extremely high, and most of them will have very low probabilities, we need efficient algorithms to reduce the enormous search space. There are numerous standard methods for doing this, and some rarely used heuristics. We implemented and tested some of them, and adapted these according to our needs. Our aim was to construct a faster method which recognized the same amount of words. The methods were tested within the framework of our segment-based recognition system, the OASIS Speech Laboratory [7, 8].

*This work was supported under the contract IKTA No. 2001/055 from the Hungarian Ministry of Education.

†Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1., Hungary,
e-mail: ghosty@rgai.inf.u-szeged.hu, kocsor@inf.u-szeged.hu, tothl@inf.u-szeged.hu

‡Department of Informatics, University of Szeged H-6720 Szeged, Arpád tér 2., Hungary,
e-mail: lfelfold@inf.u-szeged.hu

2 A Segment-based Speech Recognition Approach

In the following, the speech signal A will be treated as a chronologically increasing series of the form $a_1 a_2 \dots a_{t_{max}}$, while the set of possible phoneme-sequences will be denoted by W. Essentially the task here is to find the word $\hat{w} \in W$ defined by

$$\hat{w} = arg \max_{w \in W} P(w|A) = arg \max_{w \in W} \frac{P(A|w) \cdot P(w)}{P(A)} = arg \max_{w \in W} P(A|w) \cdot P(w),$$

where $P(w)$ is known as the *language model.*

If we optimize $P(w|A)$ directly, we use a discriminative method, while if we use Bayes' theorem and omit $P(A)$, our approach is generative. Moreover the recognition process can be frame-based or segment-based, depending on whether the model incorporates frame-based or segment-based features. The widely-used HMM is a frame-based, generative method, but in the following we will describe the recognition process in a segment-based, discriminative approach (c.f. [8]).

We assume that $P(w|A) = \prod_i P(w_i|A) = \prod_i P(w_i|A_i)$, i.e. that the phonemes are independent, and for a word $w = o_1 \dots o_l$ a phoneme o_i is based on $A_i = a_j a_{j+1} \dots a_{j+r-1}$ (an r-long segment of A, where $A = A_1 \dots A_n$). With this A_i segment, a *phoneme classifier* identifies the phoneme by some method using long-term features; in our recognition system, the OASIS Speech Laboratory, Artificial Neural Networks (*ANNs*) [3] are used, but the way the classifier actually works is of no concern to us here.

To determine the values of these $P(w_i|A_i)$ functions, we need to know the exact values of the A_is, which are determined by their start and ending times (the above j and $j + r - 1$ values). Alas, this is quite a hard task, and because automated segmentation cannot be done reliably, the program will make many segmentation hypotheses, so we must include this segmentation T in our formulae:

$$P(w|A) = \sum_T P(w, T|A) = \sum_T P(w|T, A) \cdot P(T|A) \approx \max_T P(w|T, A) \cdot P(T|A)$$

For a given T, $P(w|T, A)$ can be readily calculated with the phoneme classifier, and we handle $P(T|A)$ using ANN-s as well. For this two-class training, the elements of the "phoneme" class were marked by hand, while the others in the "anti-phoneme" class were constructed from randomly selected parts of two or more phonemes. This allows us to employ the same set of features in the segmentation procedure that was used for phoneme recognition.

3 Overview of Robust Search Methods

3.1 Definition of the search space

Before presenting the algorithms, we have to define some basic terms and notations. An array $T_n = [t_0, t_1, \dots, t_n]$ is called a *segmentation* if $0 = t_0 < t_1 < \dots < t_n \leq$

t_{max} holds, i.e. they are in increasing chronological order. We also require that every phoneme fit into some overlapping interval $[t_i, t_j]$ $(i, j \in \{0, \ldots n\}, 0 \leq i < j \leq n)$, i.e. the former speech segments are referred by their start and end times.

Given a set of words W, $Pref_k(W)$ will denote the k-long prefixes of all the words in W with at least k phonemes. Then we may construct the search tree in a recursive manner: $h_0 = (\emptyset, [t_0])$ will be the root of the tree, and $Pref_1(W) \times T_n^1$ will contain the first-level vertices. Then, for a $(o_1 o_2 \ldots o_j, [t_{i_0}, \ldots, t_{i_j}])$ leaf we link all $(o_1 o_2 \ldots o_j o_{j+1}, [t_{i_0}, \ldots, t_{i_j}, t_{i_{j+1}}]) \in Pref_{j+1}(W) \times T_n^{j+1}$ nodes.

When one or more hypothesis is discarded due to its high cost, we say that it was *pruned.*

For the algorithms, certain notations are employed. "$\leftarrow$" means that a variable is assigned a value; "$\Leftarrow$" means pushing a hypothesis into a stack. A $H(t, c, w)$ hypothesis is a triplet of time, cost and a phoneme-sequence. *Extending* a hypothesis $H(t, c, w)$ with a phoneme v and an ending time t_i results in a hypothesis $H'(t', c', w')$, where $t' = t_i$, $w' = wv$, and $c' = c + c_i$, c_i being the cost of v in an interval $[t, t_i)$. This is equivalent to $p' = p \cdot p_i$, where p', p and p_i are the probabilities of H', H, and v in an interval $[t, t_i)$, respectively, and $c_i = -ln\ p_i$. We are looking for the hypothesis with the lowest cost.

3.2 Stack decoding

The stack decoding algorithm [2] is time-asynchronous, i.e. it compares hypotheses with different ending times.

In the first step of the process we place the initial hypothesis into the stack. Then we pop the hypothesis in order to examine and extend it. Next, we put all the new hypotheses into the stack and pop the most probable of them. We repeat the process until the popped hypothesis reaches the end of the utterance.

The above algorithm works because it extends hypotheses, and their cost increases since we add these costs (non-negative real numbers) together. Thus, when we reach the end of the utterance, all unexamined hypotheses will have higher costs than our actual solution.

In practice it is common to use a finite stack. However, for large vocabularies and/or sentences (those with a huge search space) there is a danger that it will eliminate the best scoring hypotheses with a greater end time. Another problem with this method is that, by increasing the length of an utterance, the run time of the stack decoding algorithm will increase exponentially.

3.3 An A* heuristic

The A* search [4] algorithm is also a common method for finding a near-optimal solution. Here, besides the $g(H)$ value for a hypothesis H (the cost so far), there is a $h(H)$ value for estimating the cost of the remaining path. We put the hypotheses into a stack and sort them using $f(H) = g(H) + h(H)$. Basically, this is just a variation of the *stack decoding* method.

Algorithm 1 Stack decoding algorithm

```
Stack ⇐ H_0(t_0, 0, ∅)
while Stack is not empty do
  H(t_i, p, w) ← top(Stack)
  if t_i = t_max then
    return H
  end if
  for t_l = t_{i+1} ... t_max do
    for all {v | wv ∈ Pref_{1+length of w}} do
      H'(t_l, p', w') ← extend H with v on [t_i, t_l]
      Stack ⇐ H'
    end for
  end for
end while
```

Jelinek offers a method for constructing a heuristic based on examples. The exact formulae can be found in [6]. The idea behind it is simple enough. An evaluation is made on segmented, tagged data in order to calculate the average (or the minimum) cost per unit time. It should then give a good estimate of the cost for the remaining time. As regards the optimality criterion, the estimate must be not greater than the actual cost. It is quite hard to meet this criterion using the average-value based approach, but fairly straightforward to satisfy with the latter. However, when we calculate the minimum cost per unit time using the latter version, there is a certain loss of efficiency although it is still somewhat better than the simple stack decoding method. The solution might be to use some hybrid combination of the two.

Algorithm 2 Universal A* algorithm

```
Stack ⇐ H_0(t_0, 0, ∅)
while Stack is not empty do
  H(t_i, p, h, w) ← top(Stack)
  if t_i = t_max then
    return H
  end if
  for t_l = t_{i+1} ... t_max do
    for all {v | wv ∈ Pref_{1+length of w}} do
      H'(t_l, p', h', w') ← extend H with v on [t_i, t_l]
      Stack ⇐ H'
    end for
  end for
end while
```

3.4 Multi-stack decoding

This method is a time-synchronous modification of stack decoding. Instead of using just one stack (where the elements cannot truly be compared because most of them have different end times), we assign one stack for each time instance. Advancing in time, we can pop each hypothesis one at a time from the given stack, extend them, and put the new hypotheses into the right stack (which depends on their new end time) [1].

Obviously stack size is very important in this method as it can affect accuracy. Overly large stacks result in a large search space (and unnecessarily long run time), while very small stacks can prune those hypotheses whose extensions might be better at a later time. Note that, for a given stack size, the run time of the algorithm depends only on the length of the utterance (or, to be more precise, on the number of possible segments).

Algorithm 3 Multi-stack decoding algorithm

```
Stack[t_0] ⇐ H_0(t_0, 0, ∅)
for time = t_0 ... t_max do
  while not empty(Stack[time]) do
    H(t, p, w) ← top(Stack[time])
    if time = t_max then
      return H
    end if
    for t_l = time + 1 ... t_max do
      for all {v | wv ∈ Pref_{1+length of w}} do
        H'(t_l, p', w') ← extend H with v on [t_i, t_l]
        Stack[t_l] ⇐ H'
      end for
    end for
  end while
end for
```

3.5 Viterbi beam search

The standard Viterbi search algorithm is just the standard time-synchronous exhaustive search method but, as it stands, it is practically unusable. However, with a small modification it can be made rather effective. We employ a variable T called *beam width*; for each time instance t we calculate D_{min}, i.e. the lowest cost of the hypotheses with the end time t, and prune all those hypotheses whose cost D falls outside $D_{min} + T$ [5]. The value of the beam width is found by trial and error.

Several versions of this method exist. When choosing one we might use different beam widths for different end times (using greater values at the beginning of words). Or we could calculate the beam width dynamically (i.e. keeping the best N hypotheses – which is identical to the multi-stack decoding algorithm –, or

reducing the beam width when the probabilities start to decrease). In trials so far we have tested this method only with a constant beam width.

Algorithm 4 Viterbi beam search algorithm

```
Stack[t_0] ⇐ H_0(t_0, 0, ∅)
for time = t_0 ... t_max do
  while not empty(Stack[time]) do
    H(t, p, w) ← top(Stack[time])
    if time = t_max then
      return H
    end if
    for t_l = time + 1 ... t_max do
      for all {v | wv ∈ Pref_{1+length of w}} do
        H'(t_l, p', w') ← extend H with v on [time, t_l]
        Stack[t_l] ⇐ H'
        Prune Stack[t_l] with beam width T
      end for
    end for
  end while
end for
```

4 Refinement of the Multi-stack decoding algorithm

When calculating the optimal stack size for multi-stack decoding, it is readily seen that this optimum will be the one with the smallest value where no best-scoring hypothesis is discarded. But this approach obviously has one major drawback. Most of the time bad scoring hypotheses will have to be evaluated owing to the constant stack size. If we could only find a way of estimating the required stack size at each time instance, the performance of the method would markedly improve.

One possibility might be to combine multi-stack decoding with a Viterbi beam search. At each time point we keep the n best-scoring hypotheses, and discard those which are not close to the peak (thus the cost will be higher than the best cost plus the beam width). Here the beam width can also be determined empirically.

One surprising thing is that when we determine the optimal parameters (stack size and beam width) for the two methods (multi-stack and Viterbi beam), both parameters can be used together, thus making the combined search method work faster than either of them separately. We found that this worked for both test sets.

Yet another approach for improving the multi-stack method is that we can predict, at a given time instance, what stack size should be sufficient. We devised two improved methods based on this.

We trained an ANN to predict whether, at a given time instance, a bound between phonemes exists or not. Then, at each time instance, this ANN returns a

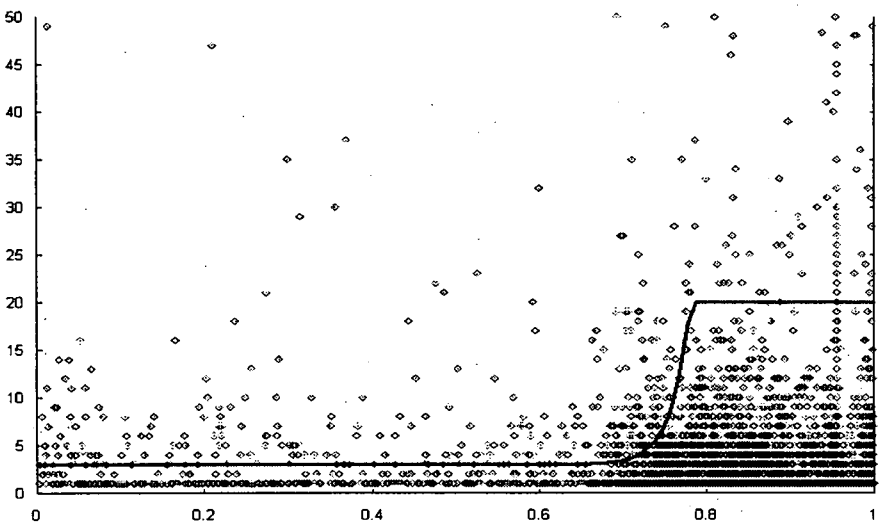

Figure 1: Bound probability – stack size diagram with the best fitting curve

probability p for this. In the first improvement we compare this p to a parameter l: if $p < l$, we use a smaller-sized stack (c_{min}), and a bigger (c_{max}) one otherwise.

We could also improve the model by fine-tuning it. To find a function that approximates the necessary stack size based on the output p of the ANN, we conducted an experiment. We recognized a set of test words using a standard multi-stack decoding algorithm with a large stack size. Then we examined the path which led to the winning hypothesis (or the first n hypotheses), and noted the required stack size and the phoneme-bound probability p at each time instance. The points of Figure 1 show the necessary stack sizes as a function of p.

For a phoneme-bound probability p (supplied by the ANN), we found that a $\min(c_0 + e^{c_1 \cdot p + c_2}, c_3)$ size stack was satisfactory. Obviously, the value for c_3 comes from the test of multi-stack decoding, and the value for c_0 from an examination of the previous improvement (as c_{min}). After, for a given c_1, c_2 can be determined by trial and error. The best fitting curve was plotted in Figure 1.

5 Experimental results

5.1 The testing sets

In trials we tested the above methods and their variations using varying parameters, namely different dictionary sizes, words, and other parameters which are method dependent (e.g. stack size in stack decoding). We also examined whether making use of a voicedetect function (which seeks to remove long, silent parts of a voice

signal) significantly improves the speed of recognition, thereby reducing the number of neuron network calls.

For this reason we created two test groups. Test set I contained only the basic elements of Hungarian numbers from six speakers. Each uttered the 26 elements twice, giving a total of 312 occurrences, while test set II contained numbers under 100 (169 test cases in all).

5.2 Results

Two things were important in the comparison. First, we had to see how good the method was in scoring correct hits. Second, the number of phoneme-classifying ANN calls made in this task. Actually, the key quantity here for evaluating a method's performance is the lowest number of ANN calls when its performance is maximal.

As the entire hypothesis space is enormous ($> 10^7$ for an average utterance) our goal is to drastically reduce it. The methods tested here require different types of parameters for optimal performance, hence they have to be listed individually.

5.3 Results of using the standard algorithms

The results of each method employed in trials are listed below.

5.3.1 Stack decoding

This method performed surprisingly well on the first test set. Extending the best-scoring of all hypotheses can be regarded as a heuristic, which performs very well with a short utterance, but on longer words it proved unsatisfactory. On the second set (whose elements were much closer to real-life examples) it yielded the worst

	hits (312)	ANN calls on set I	hits (169)	ANN calls on set II
5000	304	1,124,024	141	27,353,614
1000	304	1,124,024	139	10,278,189
500	**304**	**735,135**	137	6,798,157
250	303	661,214	136	4,135,990
100	295	562,460	**136**	**2,039,124**
50	279	500,748	127	1,148,680
25	260	354,077	124	670,369
10	210	225,152	80	281,704

Table 1: Stack decoding algorithm. The first column indicates the stack size; the best result (the one with the required accuracy and minimum ANN calls) is in **bold**.

results of all. Overall, this methods works well with short speech utterances but not with long ones. The results can be seen in Table 1.

5.3.2 Multi-stack decoding

The multi-stack decoding method seems most promising. Although it did not perform outstandingly well, it produced fair results and, unlike the other methods mentioned here (with the exception of the flexible A* algorithm) there is significant room for improvement. The main drawback of this method is the fixed stack size. Only in some cases is there a need for a maximum stack size, but here it is applied to all stacks. If we could somehow determine the stack size for each case, the performance of this method would be greatly improved. There results are shown on Table 2.

	hits (312)	ANN calls on set I	hits (169)	ANN calls on set II
100	304	8,808,675	141	7,503,876
50	304	4,421,691	141	3,719,326
25	304	2,173,794	140	1,822,171
20	**304**	**1,732,549**	138	1,449,417
15	299	1,292,938	137	1,080,198
10	295	842,595	**132**	**707,777**
5	280	416,284	119	348,066
2	240	190,994	90	155,698
1	213	119,576	59	95,938

Table 2: Multi-stack decoding algorithm. Here the parameter shown is the stack size.

5.3.3 Viterbi beam search

Of all the standard algorithms this method worked the best. On the first test set its performance ranked behind that of the stack decoding method, but on the second, more important set it performed very well, producing the lowest run times of the four standard methods. (See Table 3.)

5.4 Results of improvements

Combining standard algorithms

Among the former algorithms only the Viterbi beam and multi-stack decoding methods could be combined (the stack decoding and multi-stack decoding methods are basically different, and the A* algorithm is already an improved version of the stack decoding method). Combining the first two methods led to a more efficient

	hits (312)	ANN calls on set I	hits (169)	ANN calls on set II
25.0	304	2,032,830	141	2,806,010
20.0	304	1,223,316	139	1,394,292
19.0	**304**	**1,098,123**	138	1,211,195
18.0	303	983,711	138	1,048,396
17.0	301	884,876	137	912,880
16.0	300	790,772	135	795,547
15.0	297	704,808	**134**	**692,303**
10.0	286	380,425	128	341,587
5.0	264	201,408	98	168,941
1.0	229	131,175	71	105,807

Table 3: Viterbi beam search algorithm. Here the parameter shown is the beam width.

algorithm. This idea was included in the other improvements too. Henceforth, when we talk about improving the multi-stack decoding method, we will assume that a Viterbi beam pruning has also been applied.

Phoneme-bound detection

In order to evaluate the probability of a bound we used an ANN, which classified a bound to 80% accuracy. In the first version it achieved its goal. Acting on the first testing set the results approached those of the stack decoding results, and it performed better than the standard algorithms (see Table 4). However, on the

st_{max}	0.50	0.55	0.60	0.65	0.70	0.75	0.80
25	304	304	304	304	304	299	292
	933,993	926,151	918,376	912,275	886,313	788,429	672,395
20	304	304	304	304	**304**	299	292
	882,358	875,252	868,634	862,734	**839,789**	752,371	645,656
15	299	299	299	299	299	294	288
	788,599	782,810	777,810	772,591	750,078	684,313	594,605
10	293	293	293	293	293	288	282
	632,134	628,904	626,278	622,681	610,825	566,334	504,831

Table 4: Results using the multi-stack decoding method with the first improvement on test set I.

second set a slighter poorer result was obtained. Surprisingly, this method did slightly worse than the multi-stack decoding method with Viterbi pruning.

In the second version the e^x smoothing technique, however, worked very well.

	Set I	Set II
Stack decoding	735,135	2,039,124
A* heuristic	2,276,965	9,384,119
Multi-stack decoding	1,732,549	707,777
Viterbi beam search	1,098,123	692,303
Multi-stack decoding combined with Viterbi	922,434	474,188
Multi-stack decoding with stack size reduction I	839,789	462,363
Multi-stack decoding with stack size reduction II	749,228	427,212

Table 5: Summary of the best performances of all the methods used

On the first test set it produced almost as good a result as the stack decoding algorithm, and on the second it had the smallest run time. We can say that this novel method is definitely better than the standard algorithms. (Overall, the formula $min(3 + e^{45.0 \cdot p + 32.3}, 20)$ produced the best results.)

The best results of all methods can be seen on Table 5.

6 Conclusion

In this paper our goal was to study the search problem of speech recognition tasks, compare the standard algorithms and look for ways of improving them. Examining the test results, it is clear that we can indeed marry standard algorithms without loss of accuracy, and with a marked improvement in performance. The novel method presented here proved to be more efficient, and matched or outdid the performance of the others.

Hopefully it could be further refined by using automatic parameter determination or changing the exponential model function to some other. This will be the subject of future work.

References

[1] L.R. BAHL, P.S. GOPALAKRISHNAN, R.L. MERCER, *Search issues in large vocabulary speech recognition,* Proceedings of the 1993 IEEE Workshop on Automatic Speech Recognition, Snowbird, UT, 1993.

[2] L.R. BAHL, F. JELINEK AND R. MERCER, *A Maximum Likelihood Approach to Continuous Speech Recognition,* IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 179-190, 1983(2)

[3] C.M. BISHOP, *Neural Networks for Pattern Recognition,* Clarendon Press, Oxford, 1995.

[4] P.E. HART, N.J. NILSSON AND B. RAPHAEL, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths,* IEEE Transactions on Systems Science and Cybernetics, pp. 100-107, 1968, 4(2)

[5] P.E. HART, N.J. NILSSON AND B. RAPHAEL, *Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths",* SIGART Newsletter, No. 37, pp. 28-29, 1972.

[6] F. JELINEK, *Statistical Methods for Speech Recognition,* The MIT Press, 1997.

[7] A. KOCSOR, L. TÓTH AND A. KUBA JR., *An Overview of the Oasis Speech Recognition Project,* Proceedings of ICAI '99, pp. 95-102, Eger-Noszvaj, Hungary, 1999.

[8] L. TÓTH, A. KOCSOR AND K. KOVÁCS, A Discriminative Segmental Speech Model and its Application to Hungarian Number Recognition, *P. Sojka, I kopecek, K. Pala (eds.): TSD'2000, LNAI 1902,* pp. 307-313, 2000.