Aggregation Operators and Hypothesis Space Reductions in Speech Recognition*

Gábor Gosztolya, 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 this paper we deal with the heuristic exploration of general hypothesis spaces arising both in the HMM and segment-based approaches of speech recognition. The generated hypothesis space is a tree where we assign costs to its nodes. The tree and the costs are both generated in a top-down way where we have node extension rules and aggregation operators for the cost calculation. We introduce a special set of mean aggregation operators suitable for speech recognition tasks. Then we discuss the efficiency of some heuristic search methods like the Viterbi beam search, multi-stack decoding algorithm, and some improvements using these aggregation operators. The tests showed that this technique could significantly speed up the recognition process. The run-times we obtained were 2 times faster than the basic multi-stack decoding method, and 4 times faster than the Viterbi beam search method.

KeyWords: search methods, speech recognition, aggregation operators

In speech recognition the importance of efficient search techniques is well known. In the literature numerous improvements to speed up the search process while keeping the recognition performance constant are available [2, 3]. In an earlier paper we proposed some refinements for the well-known Viterbi beam search and the multi-stack decoding algorithm [1]. In this one we substitute the aggregation operators used for the recognition cost calculation by others that can further speed up the speech recognition process. In fuzzy theory [8] many aggregation operators are available, and we find that the family of the mean aggregation operators offers enough freedom for carrying out exhaustive trials. Out of curiosity we also introduced a special λ factor for weighting the cost values of the parameters in the mean aggregation operators so that, by advancing in time, the older cost values would be even less dominant.

The structure of the paper is as follows. First, we briefly define the probability-based approach for the speech recognition problem, the hypothesis spaces that arise, then the possible aggregation functions. Second, we discuss the basic search algorithms and the search improvements we applied to them. Finally we investigate how the aggregation operators influence both the performance and speed of the recognition system.

* This work was supported under the contract IKTA No. 2003/00056 from the Hungarian Ministry of Education.

1 The Hypothesis Space

In speech recognition problems we have a speech signal given by the series of observations $A = a_1 a_2 \dots a_t$, and the set of possible phoneme sequences (words or word sequences) which will be denoted by W. Our task is to find the word $\hat{w} \in W$ defined by

$$\hat{w} = arg \max_{w \in W} P(w|A), \tag{1}$$

which, using the Bayes' theorem, is equivalent to the following maximization problem: $\hat{w} = arg \max_{w \in W} (P(A|w) \cdot P(w))/P(A)$. Further, taking into account the fact that $P(A)$ is the same for all $w \in W$, we have that

$$\hat{w} = arg \max_{w \in W} P(A|w)P(w). \tag{2}$$

Speech recognition models can be divided into two groups (the discriminative and the generative ones), depending on whether they use Eq. (1) or Eq. (2). Throughout this paper we will apply the generative approach [2].

Unified view. Both the generative and discriminative models exploit *frame-based* and/or *segment-based* features, which allows us to have a unified view of the HMM and segment-based recognition techniques. First, we give a brief description of this scheme along with the generated hypothesis structure.

Now let us commence with some definitions. Let us define w as $o_1 o_2 \dots o_n$, where o_j is the jth phoneme of word w. Furthermore, let $A_1, A_2, \dots, A_n$ be non-overlapping segments of the observation series $A = a_1 a_2 \dots a_t$, where $A_j = a_{t_{j-1}} \dots a_{t_j}$, $j \in \{1, \dots, n\}$. An A_j segment is defined by its start and end times and is denoted by $[t_{j-1}, t_j]$. For a segmentation $A = A_1, A_2, \dots, A_n$ we collect the time indices corresponding to each segment into a vector $T_n = [t_0, t_1, \dots t_n]$ $(1 = t_0 < t_1 < \dots < t_n = t)$. We use the conventional assumption that the phonemes in a word are independent so that $P(A|w)$ can be obtained from $P(A_1|o_1), P(A_2|o_2), \dots, P(A_n|o_n)$ in some way. To calculate $P(A|w)$, various aggregation operators can be used at two distinct levels. In the first one the $P(A_j|o_j)$ probability values are supplied by a g_1 operator, i.e. $P(A_j|o_j) = g_1([t_{j-1}, t_j], o_j)$, which combines a value for measuring how well the A_j segment represents the o_j phoneme based on local information sources. In the second one another operator g_2 is used to construct the whole $P(A|w)$ probability from the $P(A_1|o_1), \dots, P(A_n|o_n)$ values.

The well-known *Hidden Markov Model* (*HMM*) [3] is basically a frame-based approach, i.e. it handles a speech signal frame by frame. Usually a *GMM* is applied to compute the $P(a_l|o_j)$ values (for delta and delta-delta features neighboring observations are also required) and for the A_j segment the $g_1([t_{j-1}, t_j], o_j)$ value is defined by $\prod_{l=t_{j-1}}^{t_j} c_{o_j} \cdot P(a_{l-k} \dots a_{l+k}|o_j)$, where $0 \leq c_{o_j} \leq 1$. Practically speaking, g_1 includes all the information we have when we are in a particular state of a HMM model. We note here that, instead of GMM, Artificial Neural Networks and other machine learning algorithms which can be used for density estimation are also viable. This provides a way for creating model hybrids. As for the $P(A|w)$ value, the g_2 operator is defined by $P(A_n|o_n) \prod_{j=1}^{n-1} (1 - c_{o_j}) P(A_j|o_j)$.

In the *segment-based speech recognition approach* – like the SUMMIT system of MIT [4] or our OASIS [5] – , g_1 will usually be the direct output of some machine learning algorithm using features that describe the whole $[t_{j-1}, t_j]$ segment. Among the many possibilities the most conventional choice of g_2 is simply to multiply the probabilities. However, later we show that using other operators is beneficial for both speed and performance.

The hypothesis space. The task of speech recognition is a selection problem over a Cartesian product space where the first dimension is a set of word hypotheses, while the second is a set of segmentations.

Given a set of words W, we use $Pref_k(W)$ to denote the k-long prefixes of all the words in W having at least k phonemes. Let

$$T^k = \{[t_0, t_1, \ldots, t_k] : 1 = t_0 < t_1 < \cdots < t_k \leq t\} \tag{3}$$

be the set of sub-segmentations made of k segments over the observation series $a_1 a_2 \ldots a_t$. The hypotheses will be object pairs, i.e. they are elements of $H = \bigcup_{k=0}^{\infty}(Pref_k(W) \times T^k)$. We will denote the root of the tree – the initial hypothesis – by $h_0 = (\emptyset, [t_0])$ ($h_0 \in H$). Here $Pref_1(W) \times T^1$ will contain the first-level nodes. For a $(o_1 o_2 \ldots o_j, [t_0, \ldots, t_j])$ leaf we link all $(o_1 o_2 \ldots o_j o_{j+1}, [t_0, \ldots, t_j, t_{j+1}]) \in Pref_{j+1}(W) \times T^{j+1}$ nodes.

Now we need to evaluate the nodes of the search tree. To this end let the g_1 and g_2 functions be defined by some aggregation operators. Then, for a node $(o_1 o_2 \ldots o_j, [t_0, \ldots, t_j])$, the value is defined by

$$g_2(g_1([t_0, t_1], o_1), \ldots, g_1([t_{j-1}, t_j], o_j)). \tag{4}$$

Note that, in practice, it is worth calculating Eq. (4) recursively. After defining the evaluation methodology we will look for a leaf with the highest probability.

2 Aggregation Operators

In this section we will first give a brief overview of mean aggregation operators, self-consistent mean operators and root-power mean operators. Then, based on these definitions, we will give a new set of aggregation operators useful for defining g_2 in the speech recognition task.

The term *mean aggregation operators* is well-known in fuzzy literature [11]. We will use the definitions of [8], but extend the terms to handle values outside the $[0, 1]$ interval. This is because, instead of a probability p, a cost $c = \log p$ value is used in practice, which induces addition instead of multiplication.

Definition 1. *A mapping $G : [0, \infty)^j \to [0, \infty)$ is called a mean aggregation operator if it satisfies the following conditions:*

M1. **Commutativity** *G is indifferent to the order of the arguments.*
M2. **Monotonicity** *$G(x_1, \ldots, x_j) \geq G(y_1, \ldots, y_j)$ if $x_i \geq y_i$ holds for $1 \leq i \leq j$.*
M3. **Idempotency** *If $x_i = c$ for all $1 \leq i \leq j$, $G(x_1, \ldots, x_j) = c$.*

Next, we need the concept of a *bag*. A bag associated with the set $[0, \infty)$ is any collection of elements drawn from $[0, \infty)$, which differs from a set in that it allows multiple copies of the same element. $\mathcal{B}^{[0,\infty)}$ will denote the set of all bags associated with the interval $[0, \infty)$. In other words, $\mathcal{B}^{[0,\infty)} = \bigcup_{j \geq 1} [0, \infty)^j$.

Definition 2. *A mapping $G : \mathcal{B}^{[0,\infty)} \to [0, \infty)$ is a self-consistent mean operator if G satisfies the following conditions:*

1. **Naturalness:** $G(x) = x$.
2. **Commutativity:** *G is indifferent to the order of the arguments.*
3. **Monotonicity:** *For bags of the same dimension condition, M2 applies.*
4. **Self-Identity:** *If $e := G(x_1, \ldots, x_j)$, then $G(x_1, \ldots, x_j, e) = G(x_1, \ldots, x_j)$.*

We will apply a special family of self-consistent mean operators – the *root-power mean operator* –, which is defined as

$$G_\alpha(x_1, \ldots, x_j) = \left(\frac{x_1^\alpha + \ldots + x_j^\alpha}{j} \right)^{\frac{1}{\alpha}}, \quad \alpha \in \mathbb{R}. \tag{5}$$

for making g_2 functions. It is well-known [9, 10], that if $\alpha \to -\infty$, $G_\alpha \to \min(x_1, \ldots, x_j)$; G_{-1} equals the harmonic mean; if $\alpha \to 0$, G_α keeps to the geometrical mean; G_1 equals the arithmetical mean; and if $\alpha \to \infty$, $G_\alpha \to \max(x_1, \ldots, x_j)$. By changing the α parameter we have a continuous transition from the minimum operator to the maximum operator.

Now let us define a variant of the root-power mean operator as

$$G_{\alpha,\lambda}(x_1, \ldots, x_j) = \left(\frac{\lambda^{j-1} x_1^\alpha + \lambda^{j-2} x_2^\alpha + \ldots + + \lambda x_{j-1}^\alpha + x_j^\alpha}{j} \right)^{\frac{1}{\alpha}}, \tag{6}$$

where $\alpha \in \mathbb{R}$ is as before and $\lambda \in [0, 1]$ is a weighting parameter. The interpretation of this operator as g_2 in the context of speech recognition is the following: x_i is the $g_1([t_{i-1}, t_i], o_i)$ value, while λ^{j-i} is a weighting factor for x_i so that advancing in time the cost of earlier phonemes will become less and less dominant in the aggregation form.

3 Search in the Hypothesis Space

Since the hypothesis space is usually huge, a full search is unfeasible. Therefore we have to use some heuristics. We chose the multi-stack decoding method and the Viterbi beam search as basic search techniques. In the following if a hypothesis is discarded (– we won't scan its descendants), we say it was *pruned*. A *stack* is a structure for keeping hypotheses in. Moreover, we use limited-sized stacks: if there are too many hypotheses in a stack, we prune the ones with the highest cost.

Multi-stack decoding method. In this algorithm we assign a separate stack to each time instance t_i and store the hypotheses in the stack according to

their end times. In the first step we place h_0 into the stack associated with the first time instance, then, advancing in time, we pop each hypothesis in turn from the given stack, extend them in every possible way, and put the new hypotheses into the stack associated with their new end times [6]. Algorithm 1 in Appendix shows the pseudocode for multi-stack decoding.

Viterbi beam search. This algorithm differs only in one feature from the multi-stack decoding approach: instead of keeping the n best hypotheses, a variable T called the *beam width* is employed. 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 hypotheses whose cost D falls outside $D_{min} + T$ [7].

Search Improvements. 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 be evaluated owing to the constant stack size. If we could find a way of estimating the required stack size associated with each time instance, the performance of the method would be significantly improved.

i) One possibility is to combine multi-stack decoding with a Viterbi beam search. At each time instance we keep only the n best-scoring hypotheses, and also discard those which are not close to the peak (thus the cost will be higher than $D_{min} + T$). Here the beam width can also be determined empirically.

ii) Another approach is based on the observation that, the later the time instance, the smaller the required stack. We attempted a simple solution for this: the stack size at time t_i will be $s \cdot m^i$, where $0 < m < 1$ and s is the size of the first stack.

iii) Another technique is a well-known modification of stacks. It can easily happen that there are two or more hypotheses which have the same phoneme-sequence and the same end times (it may be that some earlier phoneme bound is at a different time instance). In this case it is sufficient to retain only the most probable ones.

iv) Yet another approach for improving the method comes from the observation that we need big stacks only at those segment bounds where they exactly correspond to phoneme bounds. So if we could estimate at a given time instance what the probability is of this being a bound, we could then reduce the size of the hypothesis space we need to scan. We trained an ANN for this task (on derivative-like features) where its output was treated as a probability p. Then a statistical investigation was carried out to find a function that approximates the necessary stack size based on this p. First, we recognized a set of test words using a standard multi-stack decoding algorithm with a large stack. Then we examined the path which led to the winning hypothesis, and noted the required stack size and the segment bound probability p for each phoneme. The result represented as a stacksize–probability diagram was used to obtain a proper fitting curve estimating the required stack size. It can be readily shown that most of the higher stack sizes are associated with a high value of p, so the stack size can indeed be estimated by this probability.

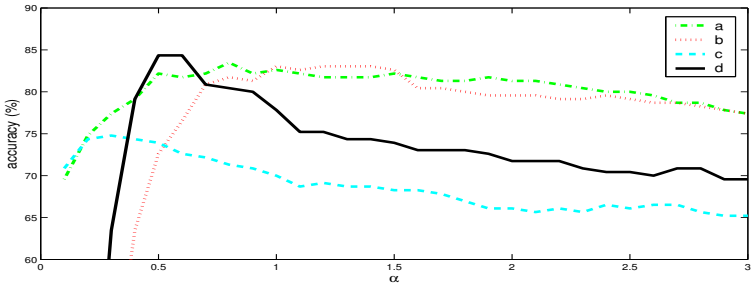

Fig. 1. Recognition accuracy for the $G_{\alpha,\gamma}$ aggregation operator ($\alpha \in [0.1, 3], \lambda = 1.0$).

4 Experiments and Results

For testing purposes we used a corpus of 500 children uttering 60 words each, making a total of 30,000 utterances of 2000 different Hungarian words with a variance related to everyday-use occurrence. Many of the children had just learned to read, which led to a diverse database. Moreover, many of the words were similar to each other with a phoneme-difference of just one or two. As a consequence, the *HTK* system scored only 84.34%.

Our aim here was to test the above search improvements – applied together with the g_2 aggregation operator – with the alpha and lambda parameters, as defined in Eq. (6). In a pilot test we found that it was worth testing only the interval $\alpha > 0$. We performed these tests using the segment-based approach. The g_1 operator was the output of a 2-layer feed-forward neural network trained on the standard segment-based features used by the SUMMIT system[4].

We also applied a modified form of g_1 and g_2: a) g_1 can be calculated in normalized form when it is multiplied by a factor depending on the length of the segment, b) instead of g_2 we may also use j times g_2 (see Eq. (6)), where j is the length of the actual word-prefix (in phonemes). This leads to four possible types (D with the $\alpha = 1$ value means conventional addition):

A) g_1 not normalized, g_2 not multiplied by j
B) g_1 not normalized, g_2 multiplied by j
C) g_1 normalized, g_2 not multiplied by j
D) g_1 normalized, g_2 multiplied by j

In the first test we examined the above aggregation methods with αs ranging from 0.1 to 3.0 with a 0.1 increments, with different λs from 0.1 to 1.0 (which means a total of $4 \times 30 \times 10$ test cases). In Figure 1 the four types can be seen with $\lambda = 1.0$. Figure 1 shows that if we don't normalize g_1, the recognition will be relatively insensitive to changes in α, but type D achieved the best results. Surprisingly $\alpha = 1.0$ usually did not produce the best results; rather the interval $[0.4, 0.7]$ seems the best for type D, and $[0.5, 2.0]$ for types A and B. The result was a recognition improvement of almost 7%.

λ	Viterbi beam search	multi-stack decoding	*iii*	*iii* + *i*	*iii* + *i* + *iv*	*iii* + *i* + *iv* + *ii*
A 1.0	41,576.98	23,699.43	14,576.20	13,577.95	11,934.11	11,911.38
B 1.0	37,764.81	12,843.03	10,544.18	8,081.93	7,153.78	7,063.16
D 1.0	67,222.28	25,571.59	14,952.41	14,926.21	11,972.71	11,179.51
A 0.9	68,789.68	22,446.94	14,552.13	14,464.07	11,840.28	11,810.74
B 0.9	41,605.54	12,840.50	10,532.37	8,138.60	7,452.19	7,411.95
D 0.9	62,080.18	19,248.69	14,962.18	13,439.83	11,392.84	10,811.37
A 0.8	84,199.83	22,446.30	17,453.82	17,172.01	15,972.26	15,625.16
B 0.8	48,809.48	16,043.51	11,696.54	9,555.46	8,943.54	8.891.57
D 0.8	62,687.10	22,448.76	17,948.40	15,460.08	14,297.17	14,072.94
A 0.7	141,165.57	38,060.62	31,522.57	31,419.86	30,683.94	30,179.25
B 0.7	56,029.24	16,042.50	11,677.73	9,612.77	9,142.64	9,098.23
D 0.7	62,182.55	22,446.83	17,943.66	16,673.19	15,179.55	15,019.66

Table 1. Average number of phoneme-extensions for different search techniques

In the second test we examined the behavior of search improvements using different α and λ values. Because an exhaustive examination would have been too involved, we restricted the λs to 0.7, 0.8, 0.9 and 1.0, and used only the α values which performed best in the first test. Then, for a fixed α, λ and aggregation type, the parameters of the search improvements were determined using the sequential forward selection technique. First we tested all the improvements one by one with optimal parameters, then we chose the one which produced the biggest speed-up. Next, we tested the remaining improvements combined with the chosen improvements, until we had gone through all the possible combinations.

Table 1 only shows the best results for each step. We expected a recognition accuracy of at least 80%. The speed is measured by average hypothesis-extensions per word: the smaller the number, the faster the algorithm is. It can be seen that a significant speed-up was achieved. (Aggregation type C could not attain the 80% value, so it was omitted from the table.)

5 Conclusion

In speech recognition, as is usual in software applications, the two key aspects are speed and accuracy. Here we suggested a new set of aggregation operators that could be used for speeding up some heuristic search methods without significantly lowering the recognition accuracy. Based on the results above, we conclude that it is worth using mean aggregation operators in speech recognition systems. In the next phase we will apply the proposed methodology to a continuous speech recognition system. This is the subject of future work.

6 Appendix

The multi-stack decoding pseudocode described by Algorithm 1. "$\leftarrow$" means that a variable is assigned a value; "$\Leftarrow$" means pushing a hypothesis into a

stack. $Stack[t_i]$ means a stack belonging to the t_i time instance. A $H(w,T)$ hypothesis denotes a phoneme sequence and time-instance sequence pair. *Extending* a hypothesis $H(w,T) = H(w,[t_0,\ldots,t_k])$ with a phoneme v and a time t_i results in a hypothesis $H'(wv, T \cup t_i) = H'(wv,[t_0,\ldots,t_k,t_i])$, where the cost of the new hypothesis is calculated via the g_2 operator, applying the g_1 function. We denote the maximal length of a phoneme by *maxlength*.

Algorithm 1 Multi-stack decoding algorithm

```
Stack[t_0] ⇐ h_0(∅, [t_0])
for i = 0 ... n do
  while not empty(Stack[t_i]) do
    H(w,T) ← top(Stack[t_i])
    if t_i = t_max then
      return H
    end if
    for t_l = t_{i+1} ··· t_{i+maxlength} do
      for all {v | wv ∈ Pref_{1+length of w}} do
        H'(w', T ∪ t_l) ← extend H with v
        Stack[t_l] ⇐ H'
      end for
    end for
  end while
end for
```

References

1. G. Gosztolya, A. Kocsor, *Improving the Multi-Stack Decoding Algorithm in a Segment-based Speech Recognizer,* Proc. of the IEA/AIE, LNAI 2718, pp. 744-749, Springer Verlag, 2003.
2. F. Jelinek, *Statistical Methods for Speech Recognition,* The MIT Press, 1997.
3. L.Rabiner, B.-H. Juang *Fundamentals of Speech Recognition* Prentice Hall, 1993.
4. J. Glass, J. Chang, M. McCandless, *A Probabilistic Framework for Features-Based Speech Recognition,* Proceedings of International Conference on Spoken Language Processing, Philadelphia, PA, pp. 2277-2280, 1996.
5. L. Tóth, A. Kocsor, K. Kovács, *A Discriminative Segmental Speech Model and Its Application to Hungarian Number Recognition, Text, Speech and Dialogue, 2000.*
6. 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.
7. P.E. Hart, N.J. Nilsson, B. Raphael, *Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths",* SIGART Newsletter, No. 37, pp. 28-29, 1972.
8. D. Dubois, H. Prade, *Fundamentals of Fuzzy Sets,* Kluwer Acad. Pub., 2000.
9. G.H. Hardy, J.E. Littlewood, G. Pólya, *Inequalities,* Cambridge Univ., 1968.
10. M.J. Cloud, B.C. Drachman, *Inequalities,* Springer, 1998.
11. E.P. Klement, R. Mesiar, E. Pap, *Triangular Norms,* Kluwer Acad., 2000.