Int J Speech Technol
DOI 10.1007/s10772-008-9005-5

The use of speed-up techniques for a speech recognizer system

András Kocsor · Gábor Gosztolya

Received: 14 December 2005 / Accepted: 9 October 2008

Abstract In speech recognition, not just the accuracy of an automatic speech recognition application is important, but also its speed. However, if we want to create a real-time speech recognizer, this requirement limits the time that is spent on searching for the best hypothesis, which can even affect the recognition accuracy. Thus the applied search method plays an important role in the speech recognition task, and so does its efficiency, i.e. how quickly it finds the uttered words. To speed up this search process, various ideas are available in the literature: we can use search heuristics, multi-pass search, or apply a family of aggregation operators. In this paper we test all these methods in turn, and combine them with a set of other novel speed-up ideas. The test results confirm that all of these techniques are valuable: using combinations of them helped make the speech recognition process over 12 times faster than the basic multi-stack decoding algorithm, and almost 11 times faster than the Viterbi beam search method.

Keywords Speech recognition · Multi-stack decoding algorithm · Viterbi beam search · Multi-pass search · Clustering algorithms · Aggregation operators

A. Kocsor · G. Gosztolya (✉)
Research Group on Artificial Intelligence, Hungarian Academy of Sciences and University of Szeged, Aradi vértanúk tere 1, 6720 Szeged, Hungary
e-mail: ggabor@inf.u-szeged.hu

A. Kocsor
e-mail: kocsor@inf.u-szeged.hu

1 Introduction

The speech recognition task is a field in artificial intelligence with high computational demands, so it is very important to make efficient use of the available CPU time. This is especially true if we would like to construct a real-time speech recognition application with a good recognition accuracy. This efficiency strongly depends on the effectiveness of the search process, and it is also heavily investigated (Ney and Ortmanss 2000; Kanthak et al. 2002), which is why this paper is devoted to this particular issue.

To speed up the search process, various ideas are available in the literature: we can do a multi-pass search (Schwartz et al. 1996), apply search heuristics (Gosztolya et al. 2003), or try out other similar ideas. However, if we were to improve the recognition accuracy by methods not requiring additional computational time, we would have more room to achieve a speed-up while maintaining the same level of accuracy. This is what we decided to do. We experimented with a family of aggregation operators (Gosztolya and Kocsor 2004; Dubois and Prade 2000) used for calculating hypothesis probabilities.

The structure of this paper is as follows. First, we define a standard speech recognition framework. Second, we describe the speed-up improvements used

Published online: 02 December 2008

along with the search process. After the experiments and test results are discussed, and then we draw some conclusions.

2 Search spaces in speech recognition

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

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

which, using Bayes' theorem, is equivalent to the following maximization problem:

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

Further, noting the fact that $P(A)$ is the same for all $w \in W$, we have that

$$\hat{w} = \arg\max_{w \in W} P(A|w) P(w). \tag{3}$$

Speech recognition models can be divided into two groups—the discriminative and generative ones—depending on whether they use (1) or (3). Throughout this paper we will apply the customary, generative approach (Jelinek 1997), but all of the tested improvements and ideas in this paper can be applied both in discriminative and generative environments.

2.1 Unified view

Both the generative and discriminative models exploit *frame-based* and/or *segment-based* (Ostendorf et al. 1996) features, and this fact allows us to have a unified framework of the Hidden Markov model and segment-based recognition techniques. We should emphasize here that this theoretical model covers all the common speech recognition models, thus the methods described in this paper are also applicable to each model. First, let us give a brief outline of this framework along with the generated hypothesis structure.

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 make 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.

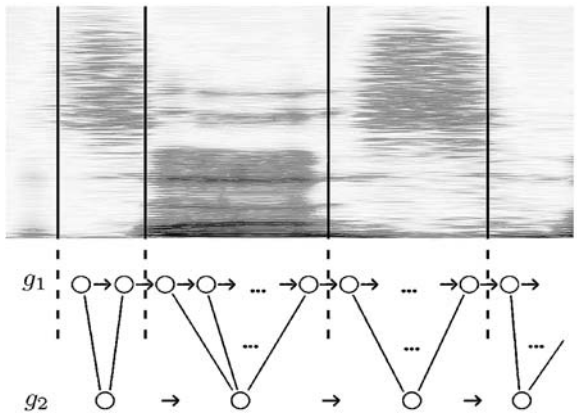

Fig. 1 Scheme of the recognition process with the two distinct levels (g_1 and g_2)

$$P(A_j|o_j) = g_1([t_{j-1}, t_j], o_j), \tag{4}$$

which provides an overall measure that tells us 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 $P(A|w)$ using the probability values $P(A_1|o_1), \dots, P(A_n|o_n)$ (see Fig. 1).

2.2 Frame-based approach

The well-known *Hidden Markov Model* (HMM) (Rabiner and Juang 1993) is basically a frame-based approach, i.e. it handles a speech signal frame by frame. Usually a *Gaussian Mixture Model* (GMM) (Duda and Hart 1973) is applied to compute the $P(a_l|o_j)$ values (for delta and delta-delta features neighbouring observations are also required) and for the A_j segment the

$g_1([t_{j-1}, t_j], o_j)$ value is defined by

$$\prod_{l=t_{j-1}}^{t_j} c_{o_j} \cdot P(a_l|o_j), \tag{5}$$

where $0 \leq c_{o_j} \leq 1$. Practically speaking, g_1 contains 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* (ANNs) (Bishop 1995) or other machine learning algorithms that 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

$$P(A_n|o_n) \prod_{j=1}^{n-1} (1 - c_{o_j}) P(A_j|o_j). \tag{6}$$

2.3 Segment-based approach

In the segment-based speech recognition approach—like the SUMMIT system of MIT (Glass et al. 1996) or our OASIS (Kocsor et al. 1999)—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. Sometimes a length normalization is also used, then this output is raised to the $(t_j - t_{j-1})$th power. Among the many possibilities the most conventional choice of g_2 is simply to multiply the probabilities, but using other operators could be beneficial for the overall performance (Gosztolya and Kocsor 2004).

2.4 The hypothesis space

The task of speech recognition is essentially 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. Now let

$$T^k = \{[t_0, t_1, \ldots, t_k] : 1 = t_0 < t_1 < \cdots < t_k \leq t\} \tag{7}$$

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

$$H = \bigcup_{k=0}^{\infty} (Pref_k(W) \times T^k). \tag{8}$$

We will denote the root of the tree—the initial hypothesis—by $h_0 = (\emptyset, [t_0])$ $(h_0 \in H)$. $Pref_1(W) \times T^1$ 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.

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

$$g_2\big(g_1([t_0, t_1], o_1), \ldots, g_1([t_{j-1}, t_j], o_j)\big). \tag{9}$$

Note that, in practice, it is worth calculating (9) recursively. After defining the evaluation methodology we will look for a leaf with the highest probability.

This definition in typical circumstances leads to a huge hypothesis space, where a full search would be impractical because of its big run time and memory requirements. This leads us to employ heuristics like the well-known Viterbi beam search (Hart et al. 1972) or our choice, the multi-stack decoding algorithm (Bahl et al. 1993), which will be described later in detail.

In this paper we employed a traditional technique to avoid underflowing: instead of a probability p, we used the cost $c = -\log p$. This will require other modifications like using addition instead of multiplication, or looking for the hypothesis with the lowest cost, and not the one with the highest probability. We applied the frame-based approach: the phoneme-level values were calculated in the way described above, but using ANN-s instead of the traditional GMM-s, and after the negative logarithm of the resulting values was calculated. Both for g_1 and g_2, the default aggregation operator adds the costs, which is equivalent to multiplying the probabilities. But later we changed both operators for the sake of increasing the recognition performance. When a hypothesis is discarded for some reason, we say that it was *pruned*.

3 Aggregation operators in speech recognition

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_1 and g_2 in the speech recognition task.

The term *mean aggregation operators* is well known in fuzzy literature (Klement et al. 2000). We

will use the definitions of (Dubois and Prade 2000), but extend the terms to handle values outside the [0, 1] interval. We need this modification because instead of a probability p we used the cost $c = -\log p$ throughout this study.

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,

$$\mathcal{B}^{[0,\infty)} = \bigcup_{j \geq 1} [0, \infty)^j. \tag{10}$$

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 with 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

$$G_\alpha(x_1, \ldots, x_j) = \left(\frac{x_1^\alpha + \cdots + x_j^\alpha}{j} \right)^{\frac{1}{\alpha}}, \quad \alpha \in \mathbb{R}, \tag{11}$$

for making g_1 and g_2 functions. It is well known (Hardy et al. 1968; Cloud and Drachman 1998), that when $\alpha \to \infty$, $G_\alpha \to \min(x_1, \ldots, x_j)$; G_{-1} equals the harmonic mean; when $\alpha \to 0$, G_α tends to the geometrical mean; G_1 equals the arithmetical mean; and when $\alpha \to \infty$, $G_\alpha \to \max(x_1, \ldots, x_j)$. By varying the α parameter we have a continuous transition from the minimum operator to the maximum operator, so this operator seems flexible enough for us to use in speech recognition tasks. For details about how we applied it, see the *Test results* section.

4 Clustering the phoneme set

Among many others, in this study we performed a multi-pass search for speed-up reasons. This multi-pass search method (see *The search process* section) is based on smaller, more compact phoneme sets, but their efficient construction is not trivial. We will describe this process in this section by defining two novel distance functions between phonemes, show that they have the right sort of properties to be metrics, and then make use of them in the phoneme-clustering problem.

There is no simple answer to the question of how we should construct the phoneme groups mentioned above, but these phoneme groupings have to be hierarchical. Thus, the phoneme sets of a phoneme grouping must be the unions of some phoneme sets of another phoneme grouping. This is necessary because of the properties of the multi-pass search technique. We might base the phoneme group construction on previous grammatical knowledge, or use the confusion matrix of the phoneme classifier. The justification of the latter option is that the recognition process is already heavily based on the phoneme classifier, so this was why we chose it.

A classifier gets some kind of features as input, and its task is to classify them into one of the $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ classes. A *confusion matrix* A is constructed in such a way that $a_{i,j}$ is the number of feature vectors belonging to ω_j from a selected test set which were classified as ω_is by the classifier (Duda et al. 2001). In our case the classifier is used to categorize the parts of speech into one of the phoneme classes. The confusion matrix of a good classifier is close to a diagonal matrix, which is why we will concentrate on the number of misclassified items (i.e. the number of examples that were incorrectly classified). A typical confusion matrix can be seen in Table 1, where the classes are the vowels of the Hungarian language.

Grouping phonemes is a standard *clustering problem* (Hand et al. 2001): some points (here, the phoneme classes) should be assigned to a certain number

Table 1 An example of a confusion matrix

	1	2	3	4	5	6	7	8	9
1	2502	3	96	35	4	0	0	4	8
2	18	965	3	24	3	0	0	0	5
3	87	8	875	19	2	0	0	5	11
4	43	11	16	271	1	1	0	1	2
5	12	2	3	2	2250	257	80	101	53
6	0	1	0	0	51	299	17	22	8
7	1	0	0	0	46	31	208	6	1
8	3	4	3	1	70	39	8	5235	111
9	7	1	6	3	19	10	2	97	461

Table 2 The normalized confusion matrix of the matrix in Table 1. The largest value of each column is noted in bold

	1	2	3	4	5	6	7	8	9
1	**0.94**	0.00	0.10	0.10	0.00	0.00	0.00	0.00	0.01
2	0.01	**0.97**	0.00	0.07	0.00	0.00	0.00	0.00	0.00
3	0.03	0.01	**0.87**	0.05	0.00	0.00	0.00	0.00	0.02
4	0.02	0.01	0.02	**0.76**	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.01	**0.92**	0.40	0.25	0.02	0.08
6	0.00	0.00	0.00	0.00	0.02	**0.47**	0.05	0.00	0.01
7	0.00	0.00	0.00	0.00	0.02	0.05	**0.66**	0.00	0.00
8	0.00	0.00	0.00	0.00	0.03	0.06	0.03	**0.96**	0.17
9	0.00	0.00	0.01	0.01	0.01	0.02	0.00	0.02	**0.70**

of *clusters* (in our case, phoneme groups). Fortunately there are some quite general algorithms for this task. The one we are going to use requires a distance function ($\mathcal{D}$) for two clusters, which will be defined below, but first we will explain how this algorithm works.

Initially each phoneme will be considered as a distinct cluster. Then, at each step, we find those C_i and C_j clusters for which $\mathcal{D}(C_i, C_j)$ is minimal, and afterwards combine them. We repeat this until $\mathcal{D}(C_i, C_j) \geq L$, where L is some parameter. (See Appendix A for the pseudocode of this algorithm.)

To define our novel distance functions first let A' be a normalized matrix for the confusion matrix A of the applied phoneme classifier (see Table 2). It takes the form

$$a'_{i,j} = \frac{a_{i,j}}{\sum_{k=1}^{K} a_{k,j}}, \quad i, j \in \{1, \ldots, K\}. \tag{12}$$

We can assume that $\sum_{k=1}^{K} a_{k,j} \neq 0$, otherwise it would mean that the jth phoneme has no examples in the test database. Next we define a distance function between phonemes based on this A' matrix. Let

$$d^1_{i,j} = \begin{cases} 0 & \text{if } i = j; \\ \infty & \text{if } a'_{i,j} = a'_{j,i} = 0 \text{ and } i \neq j; \\ -\log(a'_{i,j}) & \text{if } a'_{j,i} = 0 \text{ and } a'_{i,j} \neq 0; \\ -\log(a'_{j,i}) & \text{if } a'_{i,j} = 0 \text{ and } a'_{j,i} \neq 0; \\ \min\big(-\log(a'_{i,j}) & \\ \quad -\log(a'_{j,i})\big) & \text{otherwise,} \end{cases} \tag{13}$$

and let

$$d^2_{i,j} = \begin{cases} 0 & \text{if } i = j; \\ \infty & \text{if } a'_{i,j} = a'_{j,i} = 0 \text{ and } i \neq j; \\ -\log\left(\frac{a'_{i,j} + a'_{j,i}}{2}\right) & \text{otherwise.} \end{cases} \tag{14}$$

The idea behind defining d^1 and d^2 is to define some basic distance values between two phonemes. Ideally $d(x, y)$ should be low if the xth and yth phonemes are frequently confused by the classifier, and high if rarely so. Moreover we expect that $d(x, x)$ should be 0 and $d(x, y)$ should equal to $d(y, x)$.

Now let us define d' as a distance function between phonemes, where $d'_{i,j}$ is the length of the shortest path from i to j using the distance function of either d^1 or d^2. (We can choose either of them, but of course if we test both, this choice leads to twice as many test cases. The results can be seen in the results section.) The intention behind this step was to bring groups of similar phonemes closer together. d' is a distance function which satisfies the criteria of being a metric because:

(a) $d'_{i,i} = 0$,
(b) $d'_{i,j} = d'_{j,i}$ and
(c) $d'_{i,j} \leq d'_{i,k} + d'_{k,j}$.

Now we have to define the distance $\mathcal{D}(C_i, C_j)$, i.e. the distance of the phoneme clusters C_i and C_j, using the $d'(x_i, y_i)$ values, which are the distances between dif-

ferent phonemes. To do this we have two straightforward choices (Hand et al. 2001):

$$\mathcal{D}_{\min}(C_i, C_j) = \min_{x,y}\{d'(x, y)|x \in C_i, y \in C_j\}, \quad (15)$$

or

$$\mathcal{D}_{\max}(C_i, C_j) = \max_{x,y}\{d'(x, y)|x \in C_i, y \in C_j\}. \quad (16)$$

The former tends to create longer, larger clusters, while the latter usually creates more compact ones. In our experiments we tested both versions.

We should note here that the use of $\mathcal{D}_{\max}$ in this algorithm could lead to a non-deterministic case if, at any given point, there exist some clusters C_i, C_j and C_k such that $\mathcal{D}_{\max}(C_i, C_j) = \mathcal{D}_{\max}(C_i, C_k)$. With $\mathcal{D}_{\min}$ no such problem arises, but it is not a metric because in some cases the triangle inequality does not hold: there exist C_i, C_j and C_k clusters such that

$$\mathcal{D}_{\min}(C_i, C_j) \not\leq \mathcal{D}_{\min}(C_i, C_k) + \mathcal{D}_{\min}(C_k, C_j). \quad (17)$$

5 The search process

Given the phoneme groups—and hence the hypothesis space—we still have to search for the best hypothesis. There are standard search heuristics for this task, and the one we opted for was the multi-stack decoding algorithm. Moreover, it is possible to construct multi-pass methods where there are multiple steps in the search process. In this paper we applied this idea using the previously constructed phoneme groups.

5.1 Multi-pass search strategies

In general, *multi-pass methods* (Schwartz et al. 1996) work in two or more steps: in the first pass the less likely hypotheses are discarded because of some condition requiring low computational time. Then, in the later passes, only the remaining hypotheses are examined by more complex, reliable evaluations, which will approximate the probabilities of the hypotheses more closely. In the common search methods only the last pass remains, so more hypotheses are scanned there, making the process more time-consuming.

In a conventional two-pass procedure only one such quick check is done, then all the remaining hypotheses are examined in the final step (Chelba 2000). But there is the possibility of examining these remaining hypotheses with more sophisticated, but still quick—yet not final—techniques, and to discard those hypotheses which seem to be unlikely. This way we can construct a multi-pass method consisting of an arbitrary number of passes, which is also not uncommon in the literature (Evermann et al. 2004; Ayako et al. 2003).

To speed up these earlier steps, we need to construct faster phoneme classifiers, and the usual way of doing this is by reducing the number of features. In our system, where ANNs are used, it also leads to a lower number of hidden neurons. However, now we chose a different technique: the number of phoneme groups was decreased, thus in the first pass a search with a very restricted phoneme set was performed. Then, in the later passes, more and more detailed phoneme groupings were used, where the dictionary consisted of the most promising words of the previous level. (This meant a lower number of output neurons for ANNs, which also led to the decrease of the number of hidden ones.) Obviously, during the last pass we had to use the original phoneme set to obtain only one word as a result, not a set of words, but because of the reduced dictionary used, it can be done much faster. At each step we employed the multi-stack decoding algorithm in the search process.

5.2 The multi-stack search method

We chose the multi-stack decoding heuristics (Bahl et al. 1993) as our basic search technique. To describe the method first we have to give a definition. 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.

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 (the initial hypothesis) into the stack associated with the first time instance, then, advancing in time, we pop each hypothesis from the given stack, extend them in every possible way, and put the new hypotheses into the stack associated with their new end times. Algorithm 2 in Appendix B shows the pseudocode of multi-stack decoding.

This algorithm has one parameter, the *stack size*. Decreasing it usually reduces the accuracy of the

method (i.e. the recognition performance), while a greater stack size leads to increased run times and therefore a slower speech recognition system. Thus it is very important to find the best parameter value, which might mean a trade-off between accuracy and speed. Of course, above a certain parameter setting there is no change in accuracy and this is often what we call the optimal value.

5.3 The Viterbi beam search algorithm

This algorithm differs only in one respect 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 $C_{\min}$, i.e. the lowest cost of the hypotheses with the end time t, and prune all hypotheses whose cost C falls outside $C_{\min} + T$ (Hart et al. 1972).

5.4 The speed-up improvements

When calculating the optimal stack size for the multi-stack decoding algorithm, 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 smart way of estimating the required stack size associated with each time instance, the performance of the method would be significantly improved. We modified the stack first so as to keep only the most probable hypothesis from the ones having the same phoneme-sequence and with the same end time, which could also be regarded as an improvement.

(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 their cost is higher than $C_{\min} + T$). 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 stack required. 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. Of course m should be close to 1, otherwise the stacks would soon be far too small.

(iii) 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 the 13 MFCC Δ 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 for estimating the required stack size (Gosztolya et al. 2003). 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. This observation was clearly confirmed by the test results.

In this paper, because of the large number of different test combinations, we did not try to determine the actual curve for the stack size in each test case. Another reason was the uneasy combination of this improvement and the multi-pass search, where the effect of a stack size curve in one pass on the whole recognition process cannot be easily determined. Rather, we applied a greatly simplified form of it: if the probability of the given time instance being a bound is greater than a parameter p_0, then the stack size belonging to this time instance will be reduced to the second parameter s_0. Although it seems to be a quite primitive tool for speeding up the search algorithm, the experiments showed that it is indeed effective.

6 Test results

We tested all these improvements on an isolated word recognition task. The reason for doing this was that there were only a few databases available for Hungarian that were of a satisfactory size. In the near future we plan to carry out tests on continuous speech recognition tasks, but it is very difficult for Hungarian—in contrast with those for the English language. This is because Hungarian is an agglutinative language—which means that a word can take many forms, making

it very hard to successfully apply an n-gram language model (Szarvas et al. 2000). However, our preliminary tests suggest that the improvements are also valuable in such an environment.

The train database consisted of 400 Hungarian speakers, each uttering 10 sentences and 4 words over the telephone (Vicsi et al. 2002). The first test database contained 10 sentences and 4 words from other 100 speakers recorded over the telephone, and it was used for the phoneme-construction task. The other test database consisted of these 500 speakers uttering the name of a different town or city, and it served the task of word recognition. Some of these utterances were unrecognizable even to us, however. They were removed from the test database, so in the end it contained 431 different words. The fact that the train and (word recognition) test databases are not strictly separate because the same persons uttered the words might seem a problem at first glance, but for a database of this size it should not really matter. The vocabulary consisted of the original 500 words.

The *HTK* system (Young 1995), which is a frame-based system that applies the HMM method, was used as a reference and produced a score of 92.11% here, using the default settings of the system (i.e. MFCC $+\,\Delta + \Delta\Delta$ features, 3-state monophone models). This value may not seem that high; but we think that the properties of the test database (recorded over the phone, complicated and sometimes very similar town-names, etc.) probably account for this. On the other hand, here we were not interested in the actual recognition scores, but in the attainable advantage of speed while maintaining the original recognition performance.

The testing was done in the framework of the OASIS Speech Laboratory (Kocsor et al. 1999). The model was a frame-based one, i.e. the small, equal-sized parts of the spectrum were first classificated as phonemes by a machine learning method. The features employed were the standard 39 MFCC $+\,\Delta + \Delta\Delta$ coefficients. We used Artificial Neural Networks, but using GMMs would not have made any difference to the outcome. The minus logarithm of these scores were then aggregated into phoneme-level values by applying g_1 (addition by default, because we are dealing with costs), then the phoneme-level values were aggregated by g_2 (the default operator was also the addition operator). This way our system initially was equivalent to a HMM/ANN hybrid (Morgan and Bourland 1995) with a state transition probability of 1.0. We could do it because it was shown earlier that the state transition probabilities have practically no effect on recognition performance (Bourland et al. 1994). The baseline recognition accuracy of this system was 94.20%.

6.1 The sequence of tests

Because we used various types of speed-up techniques, we soon realized there were many different ways we could combine them. We decided that first the aggregation operators would be tested, then the best variation with the best parameter setting would be used later on. Next, the multi-stack decoding algorithm was applied with a stack size determined as the lowest one that maintained the optimal recognition accuracy. Then multi-pass tests were applied where we also looked for the combination which produced the biggest speed-up. Because we had four versions for phoneme group generation, each leading to three possibilities of recognition steps, this led to 12 variations. One more variation was the recognition in one pass with the original phoneme set, so we had 13 variations in total. In the last test the improvements were tested on these 13 variations.

6.2 Testing the aggregation operators

Previously we defined the *root-power mean operator* in (11) as

$$G_\alpha(x_1,\ldots,x_j)=\left(\frac{x_1^\alpha+\cdots+x_j^\alpha}{j}\right)^{\frac{1}{\alpha}},\quad \alpha\in\mathbb{R}. \tag{18}$$

First we will use it to construct the phoneme-level cost from the frame-level costs (g_1). We wanted to include the addition operator as a special case of some more general operator, for which we have two plausible possibilities:

$$g_1 = j\cdot G_\alpha(x_1,\ldots,x_j) \tag{19}$$

and

$$g_1 = (x_1^\alpha+\cdots+x_j^\alpha)^{\frac{1}{\alpha}}. \tag{20}$$

In this test we investigated the effect of them on the above aggregation methods with the α parameter ranging from 0.02 to 3.00 with a 0.02 increments. In Fig. 2 the two types are shown. It is readily seen that if we

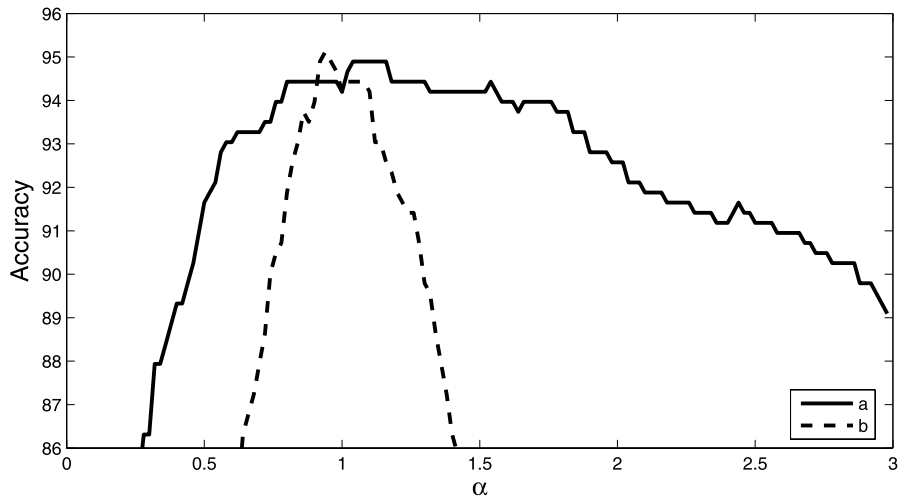

Fig. 2 Recognition accuracy using two variations of the G_α aggregation operator ($\alpha \in [0.02, 3]$) as g_1: version *a* is given in (19), while version *b* is defined in (20)

use the version described by (20) with $\alpha = 0.94$ instead of the default $\alpha = 1.00$, we can increase the recognition performance by about 1%, thus we will use this setting later on. Almost the same level of improvement can be attained with the other version (see (19)) and $\alpha = 1.06, \ldots, 1.16$. Out of curiosity we ran a test using the original $g_1 = G_\alpha(x_1, \ldots, x_j)$ version, but this test did not produce any acceptable results. Now we apply this *root-power mean operator* as g_2, as we need it to construct hypothesis-level costs from the phoneme-level costs. Both previous variations were already length-normalized, which means that the cost of a segment was based also on its length, not just on the similarity of the segment and the given phoneme (unlike in some segment-based frameworks where g_1 is the direct output of a machine learning algorithm). However, it can happen that in the final recognition, after applying g_2, a length-unnormalized version of g_1 works better. That is why we also apply a modified form of (11) in two ways. The first is that g_1 can be calculated in an unnormalized form when it is divided by the length of the segment (and will be denoted by a $'$ sign). The second option is that for g_2 we may also use j times the result of the root-power mean operator (see (11)), where j is the length of the actual word-prefix (in phonemes). This leads to four possible types:

$$g_2 = G_\alpha(x_1, \ldots, x_j), \tag{21}$$

$$g_2 = j \cdot G_\alpha(x_1, \ldots, x_j), \tag{22}$$

$$g_2 = G_\alpha(x'_1, \ldots, x'_j) \quad \text{and} \tag{23}$$

$$g_2 = j \cdot G_\alpha(x'_1, \ldots, x'_j). \tag{24}$$

Table 3 Number of phoneme groups for the various distance functions and passes. The original phoneme set ($\mathcal{P}_0$) consisted of 36 phonemes

Passes	d^1		d^2	
	$\mathcal{D}_{\min}$	$\mathcal{D}_{\max}$	$\mathcal{D}_{\min}$	$\mathcal{D}_{\max}$
$\mathcal{P}_1$	25	26	27	28
$\mathcal{P}_2$	17	16	19	17

The results can be seen in Fig. 3. Surprisingly, the results show that we were not able to improve the recognition accuracy this way: the optimal value was achieved by the version described in (23) with $\alpha = 1.00$, which means the default addition of the costs. This could be because all the possible ways of improving the accuracy had already been exploited via g_1.

6.3 Multi-pass search methods

For the multi-pass search, first we have to cluster the original phoneme set. Applying the clustering algorithm for this reason, using one of the above $\mathcal{D}$ functions will lead to a series of unions and a series of distance values. Based on them we can choose the possible values of the limit L, which will lead to the phoneme groups during the recognition process. Obviously, good L values are those where there is a nice gap between successive distance values in the output.

On examining Fig. 4 we had to look for the bigger flat regions on each curve because these represent big gaps between phoneme cluster distances. On each of them we selected two L values, resulting in the same

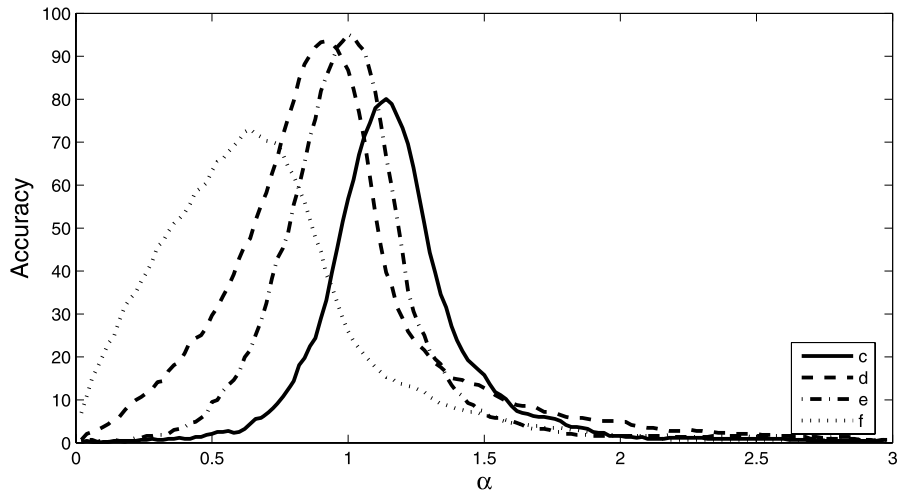

Fig. 3 Recognition accuracy using the four tested variations of the G_α aggregation operator ($\alpha \in [0.02, 3]$) as g_2: versions c, d, e and f are given in (21), (22), (23) and (24), respectively

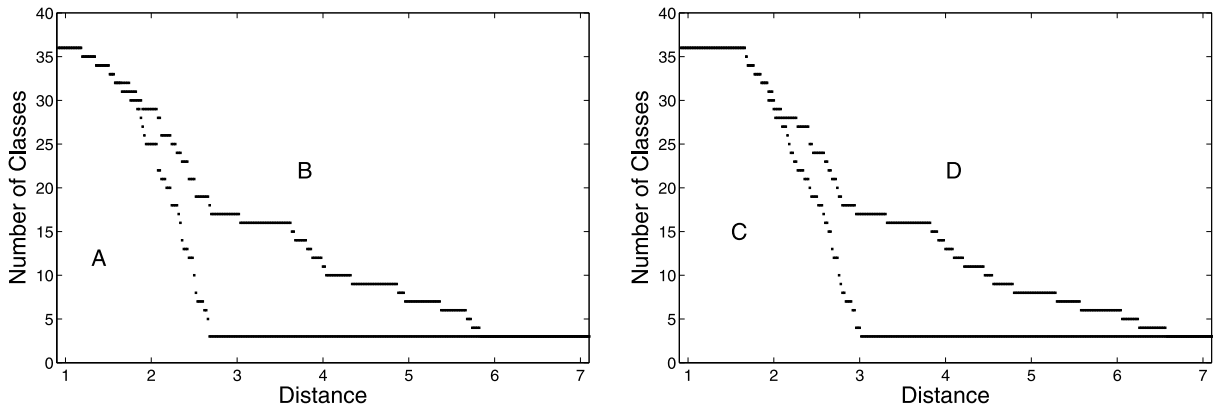

Fig. 4 Number of phoneme groups (classes)—L limit diagram for the four distance-variations d^1 and d^2, respectively. The A and C curves are for $\mathcal{D}_{\min}$, while the B and D curves are for the $\mathcal{D}_{\max}$ group distance function

number of phoneme groups, which were later used in the multi-pass recognition method. The corresponding recognition steps were called Pass 1 ($\mathcal{P}_1$) and Pass 2 ($\mathcal{P}_2$), with the number of phoneme groups varying from 25 to 28 and from 16 to 19, respectively. The default phoneme set was denoted by $\mathcal{P}_0$ and had 36 phonemes (of course, there was also a pass belonging to this phoneme group). The number of phonemes for each pass and distance function can be seen in Table 3.

After constructing the phoneme sets, we had to determine the stack size values for each recognition step (for each multi-pass combination). To do this we ran another exhaustive set of tests: for each pass we used a stack size of 25, 50, 75, 100, 125, 150, 175 or 200. For example, for a three-pass configuration it meant 512 test cases. In the end with each combination we kept those stack sizes that produced the lowest speed and had a word accuracy score of at least 94%.

6.4 Testing the speed-up improvements

Having the optimal aggregation operator we now need to test the speed-up improvements. However, their application is not trivial: we could use them in various combinations and most of them have some parameters which also affect the accuracy and speed of the search process. For this reason we used an iterative technique called *sequential forward selection* (Devijver and Kittler 1982; Schlimmer 1993). Here we apply it with a simple multi-stack decoding search on the original phoneme set. First all the improvements are tested separately, which includes the determination of their optimal parameters. Among them, in the fol-

Table 4 Performance of the basic search methods (the Viterbi beam search and the multi-stack decoding algorithm), and the applied, novel improvements and improvement combinations. The number of total and average (per word) ANN calls are linearly proportional to the running time

Method combinations	Total ANN calls	avg. ANN calls
Viterbi beam search	*83,929,336*	*194,731.64*
multi-stack decoding	*96,491,720*	*223,878.70*
multi-stack + i	**17,392,150**	**40,353.02**
multi-stack + ii	23,412,327	54,320.94
multi-stack + iii	22,117,284	51,316.20
multi-stack + i + ii	14,037,545	32,569.71
multi-stack + i + iii	**13,741,984**	**31,883.95**
multi-stack + i + iii + ii	11,905,349	27,662.62

lowing we keep the one which produced the biggest speed-up along with the given parameter value. In the next step we test the remaining improvements, until we have gone through all the possible combinations (Gosztolya and Kocsor 2004). The results can be seen in Table 4, where we separated the iterations with a horizontal line. The fastest configuration of an iteration was noted in *bold*, while the order of the improvements means the order of fine-tuning the parameters. Thus the tested configurations in an iteration differ only in the last noted improvement.

After this test we have a sequence of improvements, and an optimal parameter setting for the original phoneme set. We used this sequence for each pass of the 12 multi-pass configuration. However, the improvements usually had one or more parameters that had to be set manually again for each multi-pass configuration. It was a time-consuming task indeed, but it was the only way we could guaranteed that the results would have both the minimum recognition accuracy and the optimal speed-up.

6.5 Results

Because our goal was to speed up the dynamic recognition process with the above improvements while preserving the accuracy, the speed of a configuration was determined as the lowest one where the recognition accuracy was at least 94% (which was the original recognition percentage). The speed was measured in terms of average phoneme identifications per word with a value normalized to the pass belonging to the standard phoneme group. For the results see Table 5. In the table the symbol "•" means that we applied the given recognition pass in the configuration, while the symbol "○" means that this pass was omitted.

The first thing we noticed was that the usage of aggregation operators was successful: in the first step the recognition accuracy rose from 94.20% to 95.12% just by applying the root-power mean operator as g_1, with the modifications described above. In the second step, however, we were not able to increase this percentage any more at the word-level (g_2), despite testing numerous variations.

The multi-pass search method presented also proved to be successful: besides the same level of recognition accuracy, a multi-pass recognizer ran 12–59% faster than the basic multi-stack decoding method. It is also true that two-pass methods in general worked better than three-pass ones, and among them, the two-pass methods with $\mathcal{P}_1$ are usually better than the ones with $\mathcal{P}_2$. Since the initial phoneme set only consisted of 36 phonemes, there was no point in creating more phoneme groups. We can perform four or more pass searches, but we would need a larger phoneme set to justify this (Gosztolya and Kocsor 2005).

Next, we can compare the performance of the multi-pass methods based on the various distance functions. It seems that the ones which used $\mathcal{D}_{\min}$ worked less efficiently than those which used $\mathcal{D}_{\max}$; on the other hand, the performance of d^1 and of d^2 seem to be very similar, hence we cannot say which one is better.

Finally, let us look more closely at the improvements. If we examine Table 4, it shows that with these modifications the search process can indeed be speeded up significantly. But Table 5 also makes clear that these improvements cannot be applied together with all the multi-pass methods efficiently. Although the combined version is always at least twice as fast as the multi-pass configuration alone, the original multi-stack decoding method combined with these improvements often outperforms them. This, however, does not mean that this combination is futile in every case:

Table 5 Performance of the multi-pass search configurations, combined with the novel improvements. The • symbol means applying the given pass in the given configuration, while the ∘ symbol means that we omitted it. The speed here was also measured in the number of ANN calls per word

Phoneme group		Passes			Applied improvements			
		$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	–	i	i + iii	i + iii + ii
Standard		•	∘	∘	223,878.70	40,353.02	31,883.95	27,662.62
d^1	$\mathcal{D}_{\min}$	•	•	∘	128,065.86	27,150.05	22,014.37	21,587.46
		•	∘	•	139,095.04	89,208.51	73,886.70	70,967.27
		•	•	•	197,607.23	104,374.78	83,653.64	79,132.82
	$\mathcal{D}_{\max}$	•	•	∘	92,819.79	23,699.83	18,781.79	**17,871.53**
		•	∘	•	103,547.41	50,657.19	43,696.51	40,826.04
		•	•	•	121,225.49	59,817.09	46,534.93	44,673.19
d^2	$\mathcal{D}_{\min}$	•	•	∘	109,416.60	25,597.55	20,453.14	19,391.61
		•	∘	•	147,378.76	64,603.24	46,932.94	44,772.42
		•	•	•	177,443.28	77,849.67	62,205.58	52,837.09
	$\mathcal{D}_{\max}$	•	•	∘	110,234.28	28,802.34	25,665.77	24,149.25
		•	∘	•	93,444.29	39,500.06	32,972.24	31,842.79
		•	•	•	133,025.09	41,047.91	32,324.59	30,562.53

the two-pass search method (which is based on the distance function using d^1 and $\mathcal{D}_{\max}$), using $\mathcal{P}_1$ runs 36% faster with our improvements than the basic multi-stack decoding method in similar circumstances.

7 Conclusions

In this paper we tested several different approaches for speeding up a speech recognition system. Besides a type of multi-pass search, which was based on a novel distance function between phonemes, we applied search heuristics and some other ideas. In our studies we also used a family of aggregation operators to increase the recognition accuracy, which required no additional computational time. This way we had more room to decrease the running time of the recognition system while maintaining the same level of recognition accuracy. The results show that although it is not worth combining every multi-pass configuration with all of the improvements (i + iii + ii: see Table 5 above), our attempts were successful: the final hybrid search method was over 12 times faster than the basic multi-stack decoding algorithm, and ran almost 11 times faster than the Viterbi beam search heuristics.

Appendix A

The pseudocode of a general clustering algorithm. "$\leftarrow$" means that a variable is assigned a value. The parameters are the initial $x_1, x_2, \ldots, x_n$ points to be clustered (grouped), a $\mathcal{D}(C_i, C_j)$ distance function, and an L value for the stopping criterion.

Algorithm 1 General clustering algorithm

```
for i = 1, ..., n do
    C_i ← {x_i}
end for
while there is more than one cluster left do
    (i, j) ← arg max D(C_i, C_j) is minimal
    if D(C_i, C_j) > L then
        break
    end if
    C_i ← C_i ∪ C_j
    Remove C_j
end while
```

Appendix B

The multi-stack decoding pseudocode is described by Algorithm 2. "$\Leftarrow$" 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 using the g_2 operator, after applying the g_1 function. Here the maximal length of a phoneme is denoted by maxlength.

Algorithm 2 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

Ayako, I., Pellom, B., Cer, D., Thornton, A., Brenier, J. M., Jurafsky, D., Ward, W., & Byrne, W. (2003). Issues in recognition of Spanish-accented spontaneous English. In *Proceedings of the 2003 IEEE/ISCA workshop on spontaneous speech processing and recognition, paper MAP7*, Tokyo, Japan.

Bahl, L., Gopalakrishnan, P., & Mercer, R. (1993). Search issues in large vocabulary speech recognition. In *Proceedings of the 1993 IEEE workshop on automatic speech recognition*, Snowbird, UT.

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford: Clarendon.

Bourland, H., Konig, Y., & Morgan, N. (1994). *Remap: recursive estimation and maximization of a posteriori probabilities—application to transition-based connectionist speech recognition*. ICSI technical report TR-94-064.

Chelba, C. (2000). Exploiting syntactic structure for natural language modeling, Ph.D. thesis, Johns Hopkins University, Maryland.

Cloud, M., & Drachman, B. (1998). *Inequalities*. Berlin: Springer.

Devijver, P., & Kittler, J. (1982). *Pattern recognition, a statistical approach*. Englewood Cliffs: Prentice-Hall.

Dubois, D., & Prade, H. (2000). *Fundamentals of fuzzy sets*. Dordrecht: Kluwer Academic.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis*. New York: Wiley.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern classification*. New York: Wiley.

Evermann, G., Chan, H., Gales, M., Hain, T., Liu, X., Mrva, D., Wang, L., & Woodland, P. (2004). Development of the 2003 cu-htk conversational telephone speech transcription system. In *Proceedings of the 2004 IEEE international conference on acoustics, speech, and signal processing* (Vol. 1, pp. 249–252). Montreal, Canada.

Glass, J., Chang, J., & McCandless, M. (1996). A probabilistic framework for features-based speech recognition. In *Proceedings of the 1996 international conference on spoken language processing* (pp. 2277–2280). Philadelphia, PA.

Gosztolya, G., & Kocsor, A. (2004). Aggregation operators and hypothesis space reductions in speech recognition. In *Proceedings of the 2004 conference on text, speech and dialogue* (pp. 315–322). Brno, Czech Republic.

Gosztolya, G., & Kocsor, A. (2005). A hierarchical evaluation methodology in speech recognition. *Acta Cybernetica*, *17*, 213–224.

Gosztolya, G., Kocsor, A., Tóth, L., & Felföldi, L. (2003). Various robust search methods in a Hungarian speech recognition system. *Acta Cybernetica*, *16*, 229–240.

Hand, D., Mannila, H., & Smyth, P. (2001). *Principles of data mining*. Cambridge: MIT Press.

Hardy, G., Littlewood, J., & Pólya, G. (1968). *Inequalities*. Cambridge: Cambridge University Press.

Hart, P., Nilsson, N., & Raphael, B. (1972). Correction to a formal basis for the heuristic determination of minimal cost paths. *SIGART Newsletter*, *37*, 28–29.

Jelinek, F. (1997). *Statistical methods for speech recognition*. Cambridge: MIT Press.

Kanthak, S., Ney, H., Riley, M., & Mohri, M. (2002). A comparison of two lvr search optimization techniques. In *Proceedings of the 2002 international conference on spoken language processing* (pp. 1309–1312). Denver, CO.

Klement, E., Mesiar, R., & Pap, E. (2000). *Triangular norms*. Dordrecht: Kluwer Academic.

Kocsor, A., Tóth, L., & A. Kuba, J. (1999). An overview of the oasis speech recognition project. In *Proceedings of the 1999 international conference on applied informatics*, Eger-Noszvaj, Hungary.

Morgan, N., & Bourland, H. (1995). An introduction to hybrid hmm/connectionist continuous speech recognition. *Signal Processing Magazine*, May 1995, 1025–1028.

Ney, H., & Ortmanss, S. (2000). Progress in dynamic programming search for lvcsr. In *Proceedings of the IEEE*'88.

Ostendorf, M., Digalakis, V., & Kimball, O. A. (1996). From hmms to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *4*, 360–378.

Rabiner, L., & Juang, B.-H. (1993). *Fundamentals of speech recognition*. Englewood Cliffs: Prentice-Hall.

Schlimmer, J. C. (1993). Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the international conference on machine learning '93* (pp. 284–290).

Schwartz, R., Nguyen, L., & Makhoul, J. (1996). Multiple-pass search strategies. In *Automatic speech and speaker recognition advanced topics*, Philadelphia, PA (pp. 429–456). Dordrecht: Kluwer Academic.

Szarvas, M., Fegyó, T., Mihajlik, P., & Tatai, P. (2000). Automatic recognition of Hungarian: Theory and practice. *International Journal of Speech Technology*, *3*, 237–252.

Vicsi, K., Tóth, L., Kocsor, A., & Csirik, J. (2002). Mtba—a Hungarian telephone speech database. *Híradástechnika*, *LVII*(8).

Young, S. (1995). *The HMM Toolkit (HTK)* (Software and manual). http://htk.eng.cam.ac.uk/.