# Improving Speed and Accuracy in Automatic Speech Recognition

Gábor Gosztolya

Research Group on Artificial Intelligence

February 2010

Advisors:

Dr. János Csirik
Dr. András Kocsor

A dissertation submitted for the degree of doctor of philosophy of the University of Szeged

University of Szeged
Doctoral School in Informatics

# Preface

Even from the beginning of speech recognition technology two aspects proved to be very important, and perhaps the two most important ones. The first one was a goal: to recognize as the word or sentence spoken as accurately as possible has evidently a high focus as this is the purpose of the whole speech recognition process. The other one, however, is more like a limitation: this recognition has to be done in a limited (and in some cases in a *very* limited) amount of time, both as computational requirements and real-life milliseconds.

For these reasons in this work I decided to focus on these two special issues. First I introduce techniques to make speech recognition (and especially its search process) faster than before, but only to the degree that it preserves the recognition accuracy. Then, in the second part, I suggest some ideas which help improve the speech recognition accuracy while not affecting its speed.

Although in the field of speech recognition frame-based models and especially Hidden Markov Models are considered to be the state-of-the-art technology, this work is not solely built on the use of HMMs. Rather, it applies a more general point of view, in which frame-based models are just a special case of a speech recognition framework description, as well as segment-based ones. This more general approach serves as basis for experiments performed in our speech recognition group.

The first of my **acknowledgements** goes to my supervisor, Prof. János Csirik for supporting my research and for his helpful remarks regarding this dissertation. Secondly, I would like to thank András Kocsor for taking me on, and for all the fruitful discussions, and the number of ideas they resulted in. Thirdly I would like to thank László Tóth for all his suggestions and remarks. Next I would like to thank everybody working on the OASIS Speech Laboratory with me, especially László Felföldi, Dénes Paczolay and Kornél Kovács. I would also like to thank David Curley for examining this thesis from a linguistic perspective, and finally I would like to thank my wife, Noémi for supporting me throughout.

*Gábor Gosztolya, February 2010.*

# Notations

In a speech recognition context:

| | |
|---|---|
| $A$ ; $a_1, \ldots, a_t$ | the input speech signal |
| $W$ | the set of possible words or word sequences |
| $w$ ; $o_1, \ldots, o_n$ | a possible word or word sequence ($w \in W$) |
| $A_j$ ; $[t_{j-1}, t_j]$ | the segment of speech signal associated to the $j$th phoneme |
| $o_j$ | the $j$th phoneme of the word in question |
| $a_i$ | the $i$th frame of the speech signal |
| $t_j$ | a segment bound between phonemes |
| $T$ | a possible segmentation (a set of possible segment bounds) |
| $g_1$ | an operator supplying phoneme-level probabilities |
| $g_2$ | an operator supplying word- or hypothesis-level probabilities |
| $\mathcal{H}$ | the set of possible hypotheses |
| $h_0$ | the initial hypothesis |

In a phoneme-clustering context:

| | |
|---|---|
| $M$, $m_{i,j}$ | a confusion matrix and its elements |
| $M'$, $m'_{i,j}$ | a normalized confusion matrix and its elements |
| $d_{i,j}$ | the distance of the $i$th and the $j$th phoneme |
| $C_i$ | the $i$th cluster of phonemes |
| $\mathcal{D}(C_i, C_j)$ | the distance of the $C_i$ and $C_j$ clusters |
| $L$ | the value used by the stopping criterion |

In a fuzzy logics context:

| | |
|---|---|
| $G(x_1, \ldots, x_j)$ | a mean aggregation operator |
| $T(x, y)$ | a triangular norm |
| $S(x, y)$ | a triangular conorm |
| $f(x)$ | the additive generator of the given t-norm |
| $\phi(x)$ | the logarithmic generator of the given t-norm |

# Abbreviations

| | |
|---|---|
| ANNs | Artificial Neural Networks |
| CPU | Central Processing Unit |
| DFT | Discrete Fourier Transform |
| DTW | Dynamic Time Warping algorithm |
| FFT | Fast Fourier Transformation |
| GMM | Gaussian Mixture Model |
| HMM | Hidden Markov Model |
| LIF | Leaky Integrate-and-Fire neuron |
| MFCC | Mel-frequency Cepstral Coefficients |
| MFCC $+ \Delta + \Delta\Delta$ | MFCC feature set, its derivatives and the derivatives of derivatives |
| SVM | Support Vector Machines |
| T-norm | Triangular norm |
| T-conorm | Triangular conorm |

# List of Figures

# List of Tables

# Contents

# Chapter 1

# Introduction

*"Another visitor! Stay a while... Stay forever!"*
*Digitized speech on Commodore 64*
*at the beginning of the game Impossible Mission in 1985*

In the problem of speech recognition the task is to assign the correct word or word sequence (sentence) to a given utterance. Throughout the development of systems and methods for this task there were always two very important factors for them, regardless of the approaches and models applied in the background: speed and accuracy.

The accuracy of a speech recognition system is of course vital; after all, we perform speech recognition in order to get the correct transcript of the utterance. If this criterion is rarely satisfied, doing speech recognition makes no sense at all. On the other hand, the speed of the speech recognition system is also very important, since we want to get this result in a limited amount of time, which is usually in real-time (i.e. the whole speech recognition process has to finish within the same time as the duration of the utterance; preferably recording and processing in parallel). Speed remains an important issue, however, even when real-time recognition has already been achieved, because in a typical multi-task environment it is important not to waste unnecessary resources which other applications could make use of.

One might think that along with the development of hardware these two issues have slowly lost their importance. Actually this is not the case: although on an advanced machine it is now possible to solve a problem very quickly which once required state-of-the-art technology – and with the availability of the increasingly larger databases it can be done much more accurately –, the demands against speech recognition have also risen. The number of words expected to be recognised has grown exponentially, and attention has turned to the recognition of larger units instead of simply words: sentences, paragraphs, or even bigger units. Moreover, there is now a need to satisfactorily recognise utterances in a very noisy environment [23; 71], which of course also makes the task harder. Moreover, speech recognition applications were developed for low capacity devices (mobile phones, PDA-s, etc.), where it is trivially very important to keep both computation and memory requirements to a minimum [49; 72].

In addition to the above, there is usually a tradeoff between speed and accuracy:

most of the time requirements are derived from the fact that a huge number of potential word sequences (and their prefixes, the so-called *hypotheses*) have to be matched against the utterance in question. Therefore, if we reduce the number of scanned hypotheses, the whole process can be speeded up, but this can affect the accuracy of the whole process if we do it carelessly by unintentionally excluding a hypothesis that would lead to the correct sentence. Exploiting this tradeoff is usually straightforward as most heuristic search methods used have parameters (beam width, stack size, etc.), and by setting these appropriately we can control their behaviour.

The importance of speed and accuracy led myself to make them a central issue in this dissertation, generally concentrating on one or the other in each chapter. Knowing from experience that the search process consumes most of the CPU time needed for recognition, we decided to concentrate our efforts on this when we aimed for a speed-up. During a typical search method we examine a large number of hypotheses, which suggests two straightforward ways of making the system run faster. Both could lead to a substantial speed-up, but both could also result in a loss in accuracy. Since of course we should avoid the latter, during our investigations we always deliberately tried to introduce speed-up techniques which did not reduce the performance of the speech recognition system investigated.

The first general way of **making search faster** is to reduce the number of *possible* hypotheses by creating some new criteria; this way, during the search process, we have to consider fewer possibilities. We introduced two solutions for this: in the first one, a multi-pass search was constructed. In this traditional technique several search steps are employed, each one being more detailed than the earlier ones, but working only on the resulting hypotheses of the previous ones; this way first the most unlikely possibilities can be excluded quite quickly. Our multi-pass solution was based on a hierarchical clustering of the phoneme set.

The other solution was to restrict the set of possible positions of the speech signal where phonemes could start and/or end (called a *segmentation*), for which problem we constructed three novel algorithms. All three algorithms relied on a function which reflected the local spectral change (for which a straightforward solution was found), but they used it differently: the first one placed segmentation bounds equidistantly, but only if the function value was relatively high. The second one sought the local maxima of this function and suggested segmentation bounds at these points, while the last one suggested bounds more frequently in regions having a higher function value.

The other general way of speeding up the search is to reduce the number of hypotheses *scanned by the search heuristic applied.* As our basic search algorithm was the standard multi-stack decoding algorithm [3] (also known as *n-best search*), we focused on improving its performance by cleverly adjusting the number of hypotheses stored at a given point of the search process. Note that this algorithm is very similar to the well-known Viterbi beam search, thus our improvements most likely can be applied to this method as well.

There are actually plenty of possible ways of **improving the performance** of the speech recognition process, and examining all of them is practically impossible. This led me to focus on two areas. First I examined the process of aggregating probabilities, which is done at two distinct levels: either when constructing probabilities of whole segments as phonemes or when creating probability estimates for a hypothesis from these segment-phoneme probability pairs. These processes are usually based on the standard multiplication operation, which can be derived from the assumption of independence. But perhaps we can find a better-fitting model to replace this independence criterion, which could produce better recognition results; thus I experimented with several other similar operators. Along with these experiments we will present a novel method for modelling the fuzzy operators called triangular norms from the viewpoint of practical usability, the technology which could be used in areas different of speech recognition as well (although, of course, it has proven to be successful in this context).

The other problem we investigated for improving the accuracy of speech recognition is the so-called anti-phoneme modelling. In some approaches of speech recognition we have to distinguish between real phonemes and any other speech portion we may encounter like noise, multiple consecutive phonemes, parts of real phonemes, and so on. Since in this problem we have actual samples only for the set of correct phonemes and we have to decide on this basis, the problem is essentially that of one-class classification. Our baseline framework used a counter-example (or anti-phoneme) generator method which concatenated consecutive phonemes, and then a machine-learning method was trained for both the positive and negative examples. It worked quite well, but of course it was worthwhile examining whether we could find a better strategy. We utilized a general-purpose counter-example generator algorithm and found that it could actually perform better in this task than the baseline solutions. Moreover, in order to successfully apply this algorithm we had to make some modifications on it in order to dramatically reduce its running time, which of course could be applied in every task where this counter-example generation occurs, not just in the anti-phoneme modelling one.

Naturally in both topics we took care to make improvements which did not make the recognition process significantly slower. Calculating an operator instead of using multiplication takes practically no time compared to the other, CPU-demanding subtasks of speech recognition, while evaluating a classifier algorithm which was trained on some other data sets means absolutely no additional running time requirement. A further benefit of both these studies is that they resulted in methods which are not only good for speech recognition, but could be applied in other fields as well.

## 1.1 Summary by Chapters

The structure of this work is as follows. Chapter 1 contains a short summary of the aims of this dissertation and its key thesis points. Next, to make everything more understandable, I decided to include an overview: Chapter 2 covers the problem of speech recognition in a top-down manner, breaking up the general problem into ever smaller, more manageable parts. Note that this chapter does not contain any new results obtained by the author; its task is simply to introduce the notations used, and make the later chapters clearer and more comprehensible. Also, the methods used for comparing the performance of two speech recognition configurations, and the description of the databases used are included there.

The remaining chapters describe my work. However, every chapter refers to concepts which are well-known, but mostly outside of the field of speech recognition. As they are only matter in one chapter, I found the best way to include the definition and description of these things was in that chapter. It led to a typical chapter where first the actual subtask of speech recognition is described more accurately, then the corresponding – non-speech recognition – external concepts are clarified. Then we turn to the description of our solution for the given speech recognition subtask, using the ideas mentioned previously. Finally the test environment is described, the test results are presented and analyzed.

There is one exception from this scheme, namely Chapter 5. It describes the application of different operators in the cost (or probability) aggregation subtask of speech recognition, and while this subtask more or less remains the same, quite different operators are applied. There, instead of introducing all of them at the beginning of the chapter, I felt it better to do it just before their application, each relevant section starting with the description of a type of operator, describing its application in the given subtask, and ending with the test results and a discussion.

The order of the chapters more or less follows the chronological order of my work in the field of speech recognition. At the beginning of my research work I was interested in speeding up the speech recognition process, especially based on the multi-stack decoding search algorithm. Several ideas were implemented for this purpose, which are covered in Chapter 4. This chapter contains Thesis I/3, and is covered by publications [37; 38; 40; 42; 63]. Meanwhile, other speed-up techniques were developed: a way of constructing a multi-pass search, and new segmentation algorithms are described in Chapter 3, covering theses I/1 and I/2, covered by publications [39; 40; 44; 63].

Parallel to the speed-up efforts, I turned to the various types of aggregation operators and their application in speech recognition. This is described in Chapter 5, in chronological order. First the mean aggregation operators were applied, then I turned to the fuzzy triangular norms which were easier to apply and looked more promising. After utilizing a number of classic triangular norm families (having one parameter each), I found a way of applying the two-parameter norm and discovered that it was more tunable, thus I could use it more efficiently. Then I developed a methodology to model a triangular norm used even more precisely, which turned out to be the most successful of all norms tested.This chapter contains theses II/1 and II/2, and is covered by publications [36; 38; 40; 41; 43; 62; 63].

Finally my attention turned to the anti-phoneme task, trying to identify the amount of "phone-ness" of the speech excerpts which appear, which is described in detail in Chapter 6. It includes the basic solution for this task in the literature (i.e. using GMMs), and the basic solution in our system (i.e. generating anti-phonemes by choosing false starting and/or ending bounds, then separating the two classes by an ANN). Afterwards I made use of a general counter-example generation method for this task, but to successfully apply it I had to introduce several speed-up modifications for it.

These improvements turned out to be very effective for reducing the running-time of this algorithm, and the test results indicate that they did not affect its effectiveness as this method turned out to be the best among the three tested ones. On the other hand, the modifications introduced proved to be necessary to test a large number of parameter combinations in order to find a right parameter setting. This chapter contains Thesis II/3., and is covered by publication [35].

## 1.2 Summary by Results

In the following a thesis-like listing of the most important results of the dissertation is given. As the results can be easily divided into two groups (namely those for improving the speed, and those for improving the accuracy of a speech recognizer system), the following thesis points will be divided into these two groups as well. Their numbering also follows this scheme: the ones belonging to the first group will be denoted by I/1, I/2 and I/3, whereas the ones belonging to the second group will be denoted by II/1, II/2 and II/3. Table 1.1 shows which result is described in which publication by the author.

I/1. The author created a multi-pass search technique to speed up the search process in speech recognition. This method was based on the use of multiple, hierarchical phoneme sets, which were created by clustering the elements of the original phoneme set based on the confusion matrix of phoneme classification. Using the hierarchical property of the phoneme groups, the corresponding passes can be readily implemented as a more compact phoneme group fuses several words into one. This way the search process was significantly speeded up. This thesis is explained in detail in Section 3.3.1; it is covered by publications [39; 40; 63].

I/2. The author constructed a method for detecting the interchange of phonemes to more precisely model the bound between them. Based on it, he constructed a novel algorithm to supply a set of possible phoneme boundaries. As shown by the results, this algorithm can indeed lead to a significant speed-up over that of the standard segmentation method, which was our aim. Moreover, a careful configuration of this method also gave a noticeable improvement in the recognition accuracy along with a significant speed-up. This thesis is explained in detail in Section 3.3.2; it is covered by publication [44].

I/3. The author examined the behaviour of the multi-stack decoding algorithm and the Viterbi beam search method. These algorithms both store a number of hypotheses for each possible bound between phonemes. By introducing a number of techniques to more cleverly adjust the number of hypotheses stored, the author was able to speed up the search process along with little or no loss in the recognition accuracy. This thesis is explained in detail in Chapter 4; it is covered by publications [37; 38; 40; 42; 63].

| | [39] | [44] | [42] | [37] | [38] | [40] | [63] | [41] | [62] | [36] | [43] | [35] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I/1. | ● | | | | | ● | ● | | | | | |
| I/2. | | ● | | | | | | | | | | |
| I/3. | | | ● | ● | ● | ● | ● | | | | | |
| II/1. | | | | | ● | ● | ● | ● | ● | ● | | |
| II/2. | | | | | | | | | | | ● | |
| II/3. | | | | | | | | | | | | ● |

Table 1.1: The relation between the theses and the corresponding publications

II/1. In the search task of speech recognition the generated hypotheses are ranked by their *cost values*, usually calculated in two steps: first *phoneme-level* costs are aggregated from the *frame-level* ones, then the cost of the hypothesis is calculated from the phoneme-level values. The author applied a number of fuzzy functions at these levels: *mean aggregation operators* and *triangular norms* were tested to improve the recognition accuracy. He also modified the root-power mean operator by introducing a weighting parameter to make earlier observations less important. Furthermore, to set the two parameters of a generalized triangular norm, he turned the problem into a minimization one in a two-dimensional space and applied a global optimization method. The improvements in accuracy indicate that these operators fit the problem better than the default multiplication one. This thesis is explained in detail in Section 5.1; it is covered by publications [36; 38; 40; 41; 62; 63].

II/2. The author proposed an application-oriented method for modelling triangular norms by introducing the logarithmic generator function. Assuming that this function was piecewise linear, he introduced a technique to adequately fit it to the actual application based on the histogram of the occurring probabilities. He applied this method in the speech recognition minimization problem introduced in Thesis II/1, and showed that it can improve the performance by a greater amount than any other classical norms tested. This thesis is explained in details in Section 5.4; it is covered by publication [43].

II/3. The author applied a general counter-example generator method from the literature in the anti-phoneme problem of segment-based speech recognition. To make it possible to use he also introduced several modifications to this method, which were essential to reduce its time requirement. Doing this, he was able to markedly reduce the error rates of the speech recognition system, even when compared to other standard solutions for this problem. This thesis is explained in detail in Chapter 6; it is covered by publication [35].

# Chapter 2

# A Decomposition of the Speech Recognition Problem

*"'How does it work?' said Archchancellor Mustrum Ridcully, the Master of Unseen University. This was the kind of question that Ponder Stibbons hated almost as much as 'How much will it cost?' They were two of the hardest questions a researcher ever had to face."*
*Terry Pratchett – The Science of Discworld*

In this chapter the automatic speech recognition (ASR) problem will be described. Since this problem is a rather complex one, it is usually decomposed into several subtasks, which is the approach we shall also follow. The list of these subtasks is, of course, not an exact one, because there are steps which can be omitted, depending on the speech recognition environment. Moreover, some steps can be fused into one, or they can be divided further. This decomposition, in general, follows the way our speech recognition system is built, but as the subtasks listed here are in accordance with the literature, we will follow the same conventional decomposition approach in the dissertation.

It should be stressed that the point of this chapter is not to describe novel methods that the author introduced. Rather, it merely aims to provide an introduction to the field for the reader, where the improvements introduced by the author (which will be introduced in later chapters) can be realistically appraised. It is also important as Chapter 5 describes the application of fuzzy operators in speech recognition and some new techniques for fuzzy logic are introduced there as well. This way it may be also of interest for researchers of this area, but a knowledge of speech recognition principles will not be taken for granted, hence we will include the ASR basics in this chapter. To help the assessment of the methods proposed easier, the databases used for testing are also described here, along with the ways the performance of the methods are measured.

## 2.1 The Speech Recognition Problem in General

In essence, the goal of the speech recognition task is to design and implement a program that can transcribe the spoken input via the microphone to its written form. The aim of this is the same as that of any automatization problem: to make machines (computers) do things that currently humans have to do. But it is a rather general task, and it is beyond the current state of automatic speech recognition systems, so there are some restrictions which have to be made to make the problem feasible.

The first restriction is the language the system will use. It is important because this choice affects both the set of phonemes and the set of words used, but, of course, it is typically determined by the purpose we would like to apply the speech recognition system to. The next restriction must be the set of words allowed. Even for humans, meaning and context help a great deal in understanding the speech of other people, so we cannot expect an automatic speech recognition system to understand everything just by identifying the phonemes uttered one at a time, and then concatenating them. Moreover, if there are sentences or even bigger units of speech to recognize (and usually there are), a simulation of the structure of a sentence is also inevitable. These restrictions usually mean that speech recognition applications are restricted to a given domain (like medical dictation systems, ticket reservation by phone, etc.).

### Typical Applications

Generally there are two main groups of applications. In the first it is expected that only one person (or just a few people) will use the speech recognition system, but he (or they) will do it for a longer period of time. This gives us the opportunity to personalize the recognition process because there are techniques to help us better understand the speech of a specific person. (This point even helps humans in similar situations.) In this case the user may be asked to read a longer, set text when he uses the system for the first time; for multiple users, profiles can be created. Another simplification is that this kind of application is usually needed in offices, which can be assumed to be a place without much background noise, making recognition less difficult. The drawback is that there is usually a wide range of sentences and topics which are expected to be recognized, so a large number of sentence variations have to be considered. (However, a general application with no restrictions at all is very rare.) This kind of application is called a dictation system. Some examples can be found in [11; 29].

The other group of applications assumes that many people will use the system, each for a relative short period of time, but they will do it in a very similar way. Some common cases for these applications are that of ticket ordering, ticket reservation, a call center, or some sort of information bureau service. As one might imagine from these examples, conversations usually occur in quite a noisy environment (in a railway station, over the phone, etc.). In this case the system has to be more robust in order to handle the large number of different users and all the background noise; but the list of words and the structure of the sentences are quite restricted. This group of applications is usually called a dialogue system. Some examples can be found in [77; 88; 108].

Fortunately, these two groups of applications can be handled with a very similar technical background with slight changes in some functions and by paying more attention to some subtasks. Hence now we can turn to the (more-or-less identical) technical description of the speech recognition problem.

## 2.2 The Automatic Speech Recognition Problem

From a technical point of view the problem can be divided up in various different ways. One is based on the way we treat the relationship of the speech signal uttered and of the words (phoneme sequence) employed to fit it, where two basic approaches exist. Perhaps the more common one is that we try to model the possible distribution of the examples of the given sentence (or, at a lower level, the given phoneme). This way, since it "generates" this speech, is called the *generative approach*. The other one is the *discriminative approach*, where we try to separate the various entities (e.g. sentences, phonemes).

Another approach is related to how we model phonemes, as they are our primary building blocks. There are two main ways it can be done: most methods are based on the frame-based notion, which handles the speech signal as a series of independent, equal-sized small units called frames. These frames then can be classified as phonemes one at a time, and these neighbouring small, identical phonemes are combined to make one actual phoneme with a varying size. These phonemes can then be joined to make whole words or sentences, from which, in the end, we choose the most probable one. This scheme is based on the concept of combining multiple, consecutive frames into one phoneme. The most common example for it is the Hidden Markov Model (or **HMM**), which is the dominant model for speech recognition [50; 84].

The less commonly applied segment-based approach is based on different principles. In it, all the frames which are considered to make up a phoneme are treated as one unit, so their classification is also done together. Then the next step is the same as that was in the frame-based approach: from the phonemes words and sentences are constructed, from which we will look for the most probable one. Perhaps the most well-known example is the SUMMIT system of MIT [32], but it is also the default mode of our OASIS Speech Laboratory [65].

Both general approaches have their advantages as well as their drawbacks. As the choice of building the speech recognizer system on segment- or frame-based principles is an important question with its pros and cons, they will elaborated on in Section 2.3.

### 2.2.1 The Subtasks of Speech Recognition

The speech recognition problem is a rather difficult one as a whole. For this reason, it is usually decomposed into different subtasks, which can be performed one after another. The strength of this decomposition is that these subtasks can then be treated independently, which makes their formalization easier. Also, for each subtask different algorithms can be tried out, and their performance can be easily compared. Note that from a theoretical point of view the last statement is only true if these subtasks are independent; that is, they do not affect each other. This assumption is, of course, false, but from the practical point of view this requirement does not seem to matter, so we will follow this decomposition approach.

We perform this decomposition in a top-down way. This approach has two major advantages. First, it should help the reader to better understand the whole problem, and the function and importance of each given subtask. Second, it follows the set-up of the speech recognition framework the experiments were carried out in: the OASIS Speech Laboratory [65]. A similar approach can be found in Gordos [34]. Note that this work heavily relies on the segment-based speech recognition concept, which is sparsely represented in the literature; for it, we rely on the publications on the SUMMIT system of MIT [31; 32; 45].

### Voice Activity Detection

Speech recognition begins at the point where the input from the microphone is digitalized by the computer. The first step is to somehow divide this endless flow of speech into smaller (although still big), manageable parts. The most straightforward way of doing this is to look for silences of significant length, then cut the speech signal into parts between these silent regions. One such portion usually corresponds to a few sentences, which can still be quite long, but it is already of a manageable size. The obvious advantage of this choice is that longer pauses in a flow of speech usually mean the end of sentences (or even larger prosodic parts), so the recognition can be restricted to those of whole sentences. Another advantage of looking for silences is that it can be done rather easily by applying signal processing tools [34; 84].

There are many ways of doing it; in our experiments, a simple thresholding solution was applied. If the energy of the input rose above a given level, it was immediately regarded as speech. If, however, it fell below another (and of course, lower) level, and for a given time length it did not rise above the first parameter, it was viewed as silence, and was ignored until the first level was once again exceeded. Of course, many similar techniques can be employed, and recognition under noisy conditions could require much more sophisticated solutions.

### Decomposition of Probabilities

After the voice activity detection subtask we have a number of speech signals (i.e. the parts between the periods of silences) to recognize. Since we have to handle each signal independently of the others, and we perform exactly the same things on each of them, in the latter subtasks we can assume that we have only one speech signal $A$. Moreover, we can formalize it by stating that we are looking for the most probable word sequence (or, in some cases, just one word) given this speech signal, i.e.

$$\hat{w} = \arg\max_{w \in W} P(w|A), \tag{2.1}$$

where $W$ is the set of possible word sequences (or *sentences*). The *discriminative* approach of speech recognition makes use of this formula. Usually, however, Bayes' theorem is first applied to this equation, which leads to

$$\hat{w} = \arg\max_{w \in W} \frac{P(A|w) \cdot P(w)}{P(A)}. \tag{2.2}$$

Now we should note the fact that $P(A)$ has the same positive value for all $w \in W$. So if we omit it from our equations, the most probable word sequence $\hat{w}$ will still be the same, i.e. [34]

$$\hat{w} = \arg\max_{w \in W} P(A|w)P(w). \tag{2.3}$$

The generative approach of speech recognition makes use of Eq. (2.3), and we will follow it throughout this dissertation.

### Language Modelling

The factors $P(A|w)$ and $P(w)$ appear in Eq. (2.3), and luckily they can be treated separately, which means that their calculation (or rather approximation) can be assigned to different subtasks. The estimation for the $P(A|w)$ values is a rather complex problem, so we will deal with it later; first we shall focus on calculating the value of $P(w)$, which is also called the *language model*.

We seek to model the probability of any word sequence $w \in W$. In the common models we assume that the probability of $w$ is the product of the probability of its words, and the probability of each word $w_i$ ($w = w_1, w_2, \ldots, w_l$) depends only on this word and the preceding ones [34; 84]. This *chain-rule* means that

$$P(w) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_l|w_1, \ldots, w_{l-1}), \tag{2.4}$$

so in short

$$P(w) = \prod_{j=1}^{l} P(w_j|w_1, \ldots, w_{j-1}). \tag{2.5}$$

In practice $P(w_j)$ is calculated only from the word and the previous $n-1$ words (*n-gram modelling* method), thus

$$P(w_i) = P(w_i|w_{i-n+1}, \ldots, w_{i-1}). \tag{2.6}$$

These values can be obtained by simple word-counting for some training corpus (a set of texts). Now, having the probability of each word $w_i$ and assuming that they are independent, the probability of the word sequence $w$ can be calculated as

$$P(w) = \prod_{i=1}^{l} P(w_i). \tag{2.7}$$

Needless to say, if the recognition process is restricted to a special field of texts, then the modelling of word probabilities in Eq. (2.6) should also be based on texts from the same domain [18; 34; 50; 84].

There are two important special cases of the n-gram model, namely the 1-gram and 0-gram models. The former means that the probability of the next word depends only on that one word, thus the context is completely ignored. The probability of each word can, of course, be calculated as before, i.e. with a simple statistical investigation.

The other special case is the 0-gram model, which means that every word has exactly the same probability. It can be thought of as having no actual language model at all (except, of course, a list of the possible words).

The n-gram approach is a very rough description of the model and several techniques have been developed to improve its performance, most of them focusing on the problem of estimating the probability for never-seen n-grams, which is called smoothing [50; 55]. Another problem of this approach is that it was developed for isolating languages, so they cannot be expected to work very well for agglutinative languages like Hungarian as there are many possible forms of a word in the latter case [95]. This problem can be handled for example by building n-grams for word categories [7; 61] or for morphs [75].

### Signal Processing and Feature Extraction

Now we will concentrate on the acoustic model $P(A|w)$ from Eq. $(2.3)$. The first step of estimating this value is to transform the speech signal $A$, because now it is only at the level of speech, which contains far too much information and is in a form that cannot be easily handled. The signal processing subtask seeks to create a compact and relevant representation by extracting those pieces of information from it which are more important from a speech recognition point of view, and throwing away the rest.

Perhaps the most widely-used technique for this is *Fourier analysis* [34; 50; 84]. It decomposes the original utterance in terms of sinusoidal functions (called basis functions) that have different frequencies. Since the speech signal at this point is a discrete function, the variation called the Discrete Fourier Transform (DFT) can be applied. Usually a quicker version of it is used under the name of Fast Fourier Transformation (FFT) [34; 50; 84].

This subtask usually does not end at this point. Many methods exist that extract such properties from this spectral representation, which are quite suitable for phoneme-classification (which is done later). One good example for this is the common Mel-frequency Cepstral Coefficients (MFCC), which are derived from a cepstral representation of the sound; it relies on the Mel-scale, which has its frequency bands adjusted to human hearing. MFCC are used in a variety of systems with their derivatives and the derivatives of the derivatives (MFCC + $\Delta$ + $\Delta\Delta$ feature set for short) [50].

After this subtask $A$ can be represented as a series of feature vectors (in increasing time order) which consist of a fixed number of real values, i.e. $a_1, \ldots, a_t$; for example, when using the MFCC + $\Delta$ + $\Delta\Delta$ feature set, an $a_i$ vector consists of 39 real numbers.

### Phoneme Classification

Now Eq. (2.3) can be modified so that we look for the $\hat{w} \in W$ for which

$$\hat{w} = \arg\max_{w \in W} P(a_1, a_2, \ldots, a_t|w)P(w). \tag{2.8}$$

We still concentrate on the first factor, and we would like to know, for a given $w$, what the value of $P(a_1, a_2, \ldots, a_t|w)$ is.

Now we need to decompose the sentences: we will treat the word sequence $w$ as a phoneme-sequence $o_1, o_2, \ldots, o_n$. Next, let us divide $A$ into non-overlapping segments $A_1, \ldots, A_n$, each $A_j$ belonging to the corresponding phoneme $o_j$. (As $A = a_1 \ldots a_t$, we can define $A_j$ as $a_{t_{j-1}} \ldots a_{t_j}$ with $1 = t_0 < t_1 < \ldots < t_n = t$.) Note that at this point we made the assumption that a certain part of speech can be assigned to a phoneme, and every point of the speech signal is assigned to exactly one phoneme (sometimes called the *beads-on-a-string model*). Although this does not hold in every case, because the mouth and the tongue move continuously, causing neighbouring phonemes to affect each other, in practice this assumption does not significantly affect the speech recognition process. Also, this problem can be handled either by using different "phonemes" for the beginning, middle and end parts for any real (linguistic) phoneme, or by using a method for phoneme classification which can handle these things (such as features describing the beginning, middle or end of a phoneme in a segment-based framework).

Now, making the common assumption that the phonemes are independent, we get

$$P(A|w) = \prod_{j=1}^{n} P(A_j|o_j), \tag{2.9}$$

where the $P(A_j|o_j)$ values are obtained by utilizing some statistical machine-learning algorithm in some way [34]. In a segment-based model we directly calculate these $P(A_j|o_j)$ values via some machine-learning method using interval features (i.e. features like the mean, minimum or maximum, calculated on the whole $a_{t_{j-1}} \ldots a_{t_j}$ segment) [32; 65]. In our system Artificial Neural Networks (ANNs) [10] are used, and their corresponding output serve as estimates for the $P(o_j|A_j)$ probability values after length normalization. Then estimates of $P(A_j|o_j)$ can be readily obtained by a division by the priors (i.e. $P(o_j)$, which is trivial to calculate). Another common way of calculating the $P(o_j|A_j)$ values is the use of a *Gaussian Mixture Model* (*GMM*) [25].

For a frame-based model, the $P(A_j|o_j)$ values are not directly computed. First, for each frame, the $P(a_i|o_j)$ probabilities are determined (where $a_i \in A_j$), then they are combined to add up the probability of the $o_j$ phoneme for the whole $A_j$ part (usually by simple multiplication). This description corresponds to the one-state model, but its flexibility can be increased by decomposing the phonemes into three parts (*three-state model*). This can be treated similarly, only the $o_j$s should be interpreted as states rather than phonemes. The calculation of the $P(a_i|o_j)$ values is done in a similar way as in the segment-based approach, i.e. using some kind of machine-learning method [111].

We will discuss this part, especially the differences between the frame- and segment-based approaches, in Section 2.3.

### The Search Problem

At this point, if we consider a phoneme sequence $o_1, o_2, \ldots, o_n$, and the non-overlapping segments $A_1, A_2, \ldots, A_n$, we can calculate the probability of this pair. Further, if we have a number of such pairings, we can easily choose the most probable among them.

The problem is that there can be a very large number of these pairings, from which we need to find the optimal one, i.e. the one with the highest overall probability. This is a very important subtask as the efficiency of the search algorithm used heavily affects the speed of the whole speech recognition process [50].

There are two large groups of search techniques. The first one includes methods which guarantee that the phoneme sequence and the boundaries between these phonemes (i.e. the segments) which it returns are indeed the optimal ones. Perhaps the best-known such algorithms are the Viterbi algorithm (not to be confused with the Viterbi beam search heuristics) or in other terminology, the method of time-synchronous search [79]. However, they are rarely used in practical applications as they appear to be very slow, which calls for some kind of pruning heuristic.

The other group of methods is the set of heuristics. Their common feature is that they do not *necessarily* supply the optimal word sequence and the boundaries between the phonemes, but they can be quite fast compared to the methods in the first group. Also, typically we are just interested in the word sequence, which means that it does not matter if the returned pair differs from the optimal one in the mapping of phonemes to speech segments (as long as the phoneme/word sequence is the same, of course), which happens quite frequently with search heuristics.

The most common search heuristics used are the Viterbi beam search method [48] and the multi-stack decoding algorithm (also known as *n-best search*) [3]. They are very similar in the way they work, and produce quite similar results. We chose the latter as our basic technique, so in our experiments we use this one unless otherwise stated.

In chapters $3$ and $4$ we examine search algorithms in more detail, clearly define their mechanisms, and introduce techniques for the speed-up of standard methods.

### Segmentation Algorithms

At this point we are ready to do speech recognition. Thus, the remaining subtasks we describe are optional ones, and many speech recognizer systems (mostly frame-based ones, though) do not use them directly. (On the other hand, using an equidistant division as most segment-based systems do, or allowing every single frame to be a bound between phonemes, which is utilized by practically every frame-based system can be considered as employing a special segmentation algorithm.) The reason we find it useful to include them here is that some of our tests were done in a segment-based environment where these subtasks can have an important role. Also, we will describe algorithms we applied and developed for this subtask in chapters $3$ and $6$.

We are looking for the optimal pairing of the phonemes $o_1, o_2, \ldots, o_n$ and of the speech segments associated with these phonemes $(A_1, A_2, \ldots, A_n)$ for the given utterance $A$ and the set of words $W$. But it can be helpful if we further limit the possible set of these pairings, naturally only up to a point where it does not affect the recognition performance. Luckily, as we mentioned previously, in most applications we do not mind if some boundaries between phonemes differ from the optimal ones, as long as the phoneme sequences (and thus the words) match. So now we will concentrate on the choice of the $A_1, \ldots, A_n$ speech segments.

As we said earlier, the segments $A_1, \ldots, A_n$ can be referred to by their boundary elements $I = [t_0, t_1, \ldots, t_n]$, where an $A_j$ segment is equal to $a_{t_{j-1}} \ldots a_{t_j}$. (Moreover, instead of $A_j$, we can simply write $[t_{j-1}, t_j]$.) Up to this point, $I$ could take any value besides the trivial restrictions, so $t_j$ is an integer, $t_0 = 1$, $t_j < t_{j+1}$, and $t_n = t$. If we employ this subtask, then now just the elements of some set $T$ are allowed; thus we set up a new restriction: $t_j \in T$. This $T$ set is called the set of possible segmentation bounds (or *possible segmentation* for short), while the algorithm which supplies these values for the given speech signal $A$ is called the *segmentation algorithm* [32; 70]. (In the SUMMIT system the elements of $T$ are called *landmarks*.)

The main purpose of supplying these kinds of segmentations is that they reduce the number of possible pairings (i.e. hypotheses); and if there are fewer hypotheses, then it is easier, and thus faster to find the best one [87]. Naturally a badly working segmentation algorithm can ruin the performance of a speech recognition system because it can easily exclude good pairings by not allowing a phoneme boundary near a necessary position. Segmentation is a well-known problem for other fields of Artificial Intelligence (like image processing [58]), but it is rarely applied to speech recognition. In Chapter 3 we shall construct some algorithms like this for speeding up the recognition process.

### Anti-Phoneme Modelling

The other optional subtask is also commonly used in a segment-based context. In it we examine the probability estimate $P(A|w)$ further by introducing the correct segmentation of the signal into the formulae; as it is not known, it appears as a hidden variable $S$ in the segment-based formulation, the proper value of which has to be found. That is,

$$P(A|w) = \arg\max_{s \in S} P(A, s|w). \tag{2.10}$$

There are several ways of decomposing $P(A, s|w)$ further, depending on our modelling assumptions. What is common in all the derivations is that they trace the global probability back to the probabilities associated with the segments. The segments are usually assumed to be independent, and hence the corresponding local probability values will simply be multiplied. For example, Glass et al. employ the formula [31]

$$\prod_{j=1}^{n} \frac{P(A_j|o_j)}{P(A_j|\alpha)} P(s_j|o_j), \tag{2.11}$$

where $P(s_j|o_j)$ is a duration model, $A_j$ denotes the acoustic feature set extracted from the $j$th segment and $\alpha$ denotes the "anti-phoneme": a unit which covers all the possible signal samples that do not correspond to a real phoneme. Tóth et al. propose the formula [100]

$$\prod_{j=1}^{n} \frac{P(o_j|A_j)P(\overline{\alpha}|A_j)}{P(o_j)}, \tag{2.12}$$

where $P(\overline{\alpha}|A_j)$ denotes the probability that the given segment is *not* an anti-phoneme.

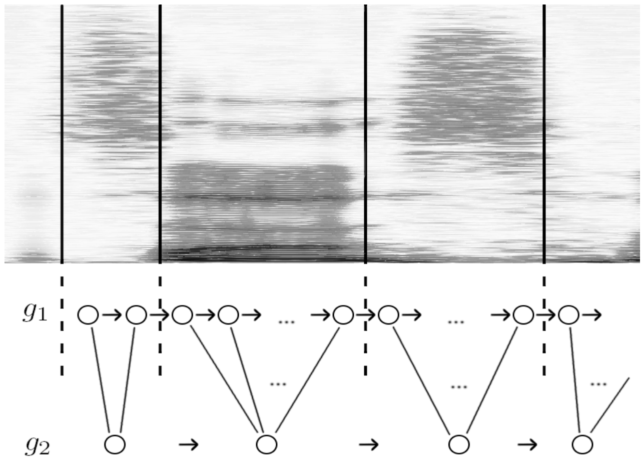

Figure 2.1: The scheme for the recognition process with $g_1$ and $g_2$.

What is interesting about this problem is that we have a large number of examples for correct segments in the form of a hand-labelled corpus; on the other hand, there is no sure way of creating anti-phonemes since these should contain anything which could occur in a sound recording that is not a correct phoneme (various kinds of noise, segments longer or shorter than one phoneme, etc.). These types of problems are called one-class classification in the field of Artificial Intelligence, and there are various strategies available to handle them. We shall examine this issue more thoroughly in Chapter 6.

## 2.3 The Segment- and Frame-Based Approaches

The choice of using a segment- or a frame-based approach is a very important one, because it can heavily influence the internal mechanisms of the speech recognition process within the various subtasks. As we perform experiments in both environments, we should describe both in more detail.

### 2.3.1 A Unified View

Both the generative and discriminative models exploit *frame-based* and/or *segment-based* [82] features, and this fact allows us to have a unified framework of the frame- and segment-based recognition techniques [82; 99]. We should emphasize here that this theoretical model covers all the common speech recognition models, thus most methods described in this dissertation are applicable to both approaches.

First let us focus on the phoneme classification subtask. Here, given an $A_j = [t_{j-1}, t_j]$ segment and an $o_j$ phoneme, we want to calculate the probability $P(A_j|o_j)$. Since this task is handled very differently in a segment- and a frame-based environment, we will generally say that these values are supplied by a $g_1$ function (we may also refer to it also as an *operator*), i.e.

$$P(A_j|o_j) = g_1([t_{j-1}, t_j], o_j), \tag{2.13}$$

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 step, another operator ($g_2$) is employed to construct the whole $P(A|w)$ using the probability values $P(A_1|o_1), \ldots, P(A_n|o_n)$, i.e.

$$P(A|w) = g_2\big(P(A_1|o_1), P(A_2|o_2), \ldots, P(A_n|o_n)\big), \tag{2.14}$$

so finally

$$P(A|w) = g_2\big(g_1([t_0, t_1], o_1),\ g_1([t_1, t_2], o_2),\ \ldots,\ g_1([t_{n-1}, t_n], o_n)\big) \tag{2.15}$$

(see Figure 2.1). There are two main reasons for introducing operators $g_1$ and $g_2$. First, it is easier to treat the frame- and segment-based approaches using this notation, and we will need them in this dissertation as our experiments are carried out in both environments. Second, although there is a default value for both operators (and in most cases these are employed), in Chapter 5 we will modify their behaviour in order to get more accurate recognition results.

### 2.3.2 The Segment-Based Approach

In the segment-based speech recognition approach – like the SUMMIT system of MIT [32] or our OASIS [65] – $g_1$ will usually be the direct output of some machine learning algorithm like GMM or ANNs. To be able to do this, we will need features which describe the whole $[t_{j-1}, t_j]$ segment. (These are typically averages of basic features on the whole segment, or on some specific part of it like the beginning, end or the middle.) Sometimes length normalization is also applied on the output of the machine learning method, which means raising it to the $(t_j - t_{j-1})$*th* power. As for $g_2$, among the many possibilities the conventional choice is simply to multiply the probabilities. Figure 2.2 shows how this approach works.

The obvious advantage of this approach is that it usually yields better phoneme classification percentages as the whole $[t_{j-1}, t_j]$ segment can be considered throughout this process [31]. The disadvantage is its speed: even a small change in one of the boundaries means that the whole classification process has to be performed again [99].

### 2.3.3 The Frame-Based Approach

The well-known *Hidden Markov Model* (*HMM*) [50; 84] is a typical frame-based method, which is used very widely (for example in [85; 109; 111]), hence it is perhaps best to describe the frame-based approach using it. Note that, for the sake of simplicity, we

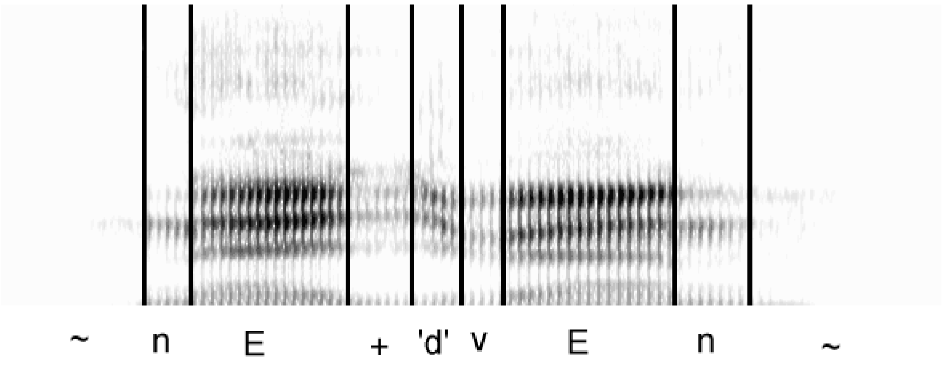

Figure 2.2: The scheme of segment-based speech recognition. The phonemes of the Hungarian word "negyven" (meaning forty) are assigned to the corresponding segments of the spectral representation of an utterance.

now deal with one-state HMM; the common solution is the three-state HMM, where each phoneme is divided into three "phonemes" (the beginning, middle and ending of the original phoneme), but it can be treated in a similar way.

The term "frame-based" simply means that speech signal is handled on a frame-by-frame basis. So first we shall process each small frame of the speech signal to see how well it represents the corresponding phoneme (i.e. we calculate the $P(a_l|o_j)$ values), which is usually done by using a *Gaussian Mixture Model* (*GMM*) [25]. Then these frame-level values of an $A_j$ segment (and thus one $o_j$ phoneme) are aggregated to one probability; i.e. the $g_1(A_j|o_j) = g_1([t_{j-1}, t_j]|o_j)$ value is defined by

$$c_{o_j}^{t_j - t_{j-1}} \cdot \prod_{l=t_{j-1}}^{t_j} P(a_l|o_j), \tag{2.16}$$

where the $c_{o_j}$ value is a *state self-transition probability* ($0 \leq c_{o_j} \leq 1$). Below Figure 2.3 shows how this frame-based model works. Instead of GMM, Artificial Neural Networks (*ANN*s) [10] or any other machine learning algorithm that can be used for density estimation is also viable, which provides a way of creating model hybrids (like the HMM/ANN hybrid [76]). As for the $P(A|w)$ value, the $g_2$ operator is simply defined by

$$\prod_{j=1}^{n-1}(1 - c_{o_j}) \cdot \prod_{j=1}^{n} P(A_j|o_j). \tag{2.17}$$

As several authors have reported that the state transition values have practically no influence on performance, they can be omitted – which is just what we will do in the following when performing experiments in a frame-based context [12; 13; 98].

The weakness of this approach lies in its accuracy: as the phoneme has to be identified from a very small piece of the speech signal, the result cannot be very precise. Moreover, the model implicitly assumes that all the frames are independent, which is a very unrealistic assumption (although it leads to a nice mathematical model) [112].

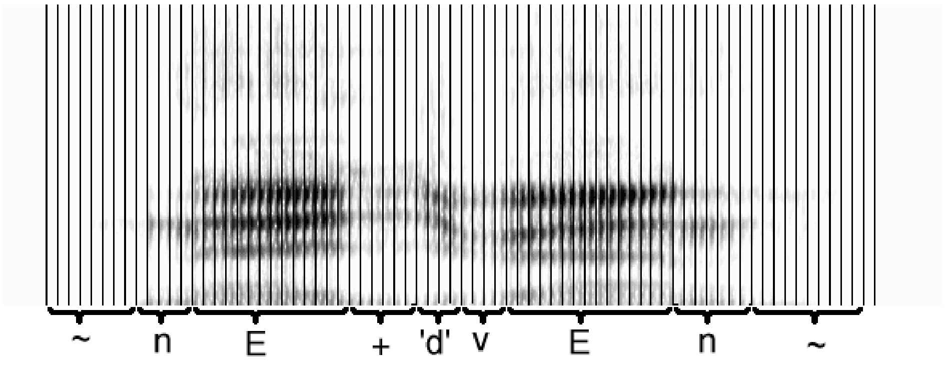

Figure 2.3: The scheme of frame-based speech recognition with the same utterance and phonemes as in Figure 2.2. Each phoneme is assigned to the spectral representation of the utterance, frame by frame.

Another disadvantage is its inability to distinguish between long and short phoneme pairs, which can be easily performed in segment-based systems (although utilizing a phoneme duration model could help in the frame-based case as well [83; 98]).

On the other hand, the "classification" of a segment is simply performed by using a combination of the frame-level values, thus it can be done extremely quickly. Also, it was noted [99] that the placement of the bounds between the segments is quite flexible in this approach. Owing to all these factors it is hard to tell which approach is better, or which should be chosen in different circumstances. Most systems, however, utilize the frame-based one, and usually implement a Hidden Markov Model [84].

## 2.4 Measurements of Performance

In practically every test we eventually have to compare the performance of two (or more) speech recognition systems, several algorithms for some speech recognition subtask, or some configuration (with different parameters, etc). The question which arises at such times is: which one works better? But "better" is not a well-defined term, so we have to come up with some concrete definitions for it. As we shall see, a configuration can be considered better than another from one of several views: it may be more accurate, it may be faster, or it may be closer to some ideal result. In the following we will describe the methodology that we will apply later on when making these kinds of comparisons.

### 2.4.1 Measurement of Accuracy

The first, and perhaps most important factor concerns how accurate a given speech recognizer configuration can be. After all, the primary goal of speech recognition is to assign the correct word sequence to the uttered speech signal. In practice it is very hard to create a recognizer that never makes a mistake, so we need to perform measurements which tell us exactly how far we are from the ideal speech recognition system. For it, we have some quite straightforward choices.

The performance of a speech recognition system can be easily measured on isolated word recognition tasks: we only have to compute the ratio of the correctly recognized and the tested words. In some tests in this dissertation we apply this technique. However, if we are performing sentence recognition, we cannot simply compare the original and the resultant sentences, because this way even one badly identified word would ruin the whole sentence. (This is also true for units larger than sentences like continuously spoken paragraphs. But from our point of view these can also be considered as sentences as both are word sequences.) We cannot compare the two sentences word for a word either, because one incorrectly inserted or omitted word – a mistake which happens quite frequently – would also ruin the calculated performance ratio. For this reason, usually the edit distance of the two sentences (the original and the resultant) is calculated; that is, we construct the resulting sentence from the original using the following operations: inserting and deleting a word, and substituting one word with another one. Next we assign some cost these operations (we use the common values of 3, 3 and 4, respectively), then we choose an operation series with the lowest cost. Finally we can use these values to calculate the following measures:

$$\text{Correctness} = \frac{N - S - D}{N} \tag{2.18}$$

and

$$\text{Accuracy} = \frac{N - S - D - I}{N}, \tag{2.19}$$

where $N$ is the total number of words in all the original sentences, $S$ is the number of substitutions, $D$ is the number of deletions and $I$ is the number of insertions [111]. We will generally employ these two formulas throughout our tests.

Note that, out of the two, accuracy is the more important one, because it deals with insertions, which are completely ignored by correctness. The reason why we always include correctness with accuracy is that it is frequently done so in the literature.

It is also common to focus on the *error rate* instead of the accuracy and/or correctness value; that is, instead of talking about a $95.03\%$ accuracy score we say that it had a $4.97\%$ of error rate. An improvement in the accuracy score also means a decrease in the corresponding error value; in this example, if the accuracy score rose to $96.55\%$, we could also say that the error rate decreased to $3.45\%$. In this case it is also common to speak of the *relative error reduction* ratio; i.e. how much of the error was eliminated in percentage terms. In this example we got a relative error reduction (or *RER*) score of $30.58\%$.

### 2.4.2 Measurement of Speed

The other important factor of a speech recognition configuration is its speed. It is a natural expectation that a speech recognition system should work in real-time; i.e. the complete processing of a sentence should not take longer than it took to pronounce it, otherwise the user will have to wait before he can start uttering another sentence. Even if this requirement is satisfied, speed still remains an important factor, because in a multi-task environment it is expected that we use as small a percentage of the available CPU as possible. (In a single-task environment, of course it is enough if we stay under $100\%$.) Because several of our investigations focus on speed and speed-ups, it is important that we measure the overall computational speed as accurately as possible.

In our experiments we confirmed the well-known fact that the search part consumes most of the CPU time used. Voice activity detection is a rather easy part, and it has to be applied only at the beginning of the recognition process. Although the signal processing and feature extraction subtask requires a great deal of mathematical computations, they are also done only once for a given utterance, and they are not affected by the demand of ever-growing vocabulary size. The phoneme classification and language modelling subtasks do require valuable CPU-time; but it is the search process which decides that how many times these are executed.

(Notice that, especially in a frame-based context, it is possible to perform phoneme classification only once for each possible phoneme, and the result is then stored in a cache. In this case it is true that phoneme classification will occur quite rarely; but then the actual classification will be overwhelmed by the thousands of cache-queries. So in the end the running time will be quasi-linear again to the phoneme classification *queries*, which will depend on the search subtask.)

For this reason we usually measured the speed of a speech recognizer configuration based on the number of phoneme classifications. We did this because it was easier to implement and to measure, and it was easier to guarantee its reliability. As we were interested only in the speed-up percentage scores, we expressed the running speed of the improved recognition configuration as a *reciprocal rate* of time consumption ratio of the baseline score. Thus a speech recognizer configuration consuming the time of one-third of the baseline is said to have achieved a speed-up rate of $300\%$, $100\%$ being the baseline.

### 2.4.3 Measurement of a Segmentation

In this dissertation we shall also deal with different segmentation algorithms, which means that we will have to measure the correctness of a possible segmentation that each produces. There are many ways of doing this; perhaps the easiest one is to compare the segmentation $T$ with the actual phoneme bounds. To do this, first the elements of $T$ and the actual phoneme bounds (the elements of $T_I$) have to be mapped to each other somehow. After this step one can easily see how many boundaries are missing from $T$ and how many of its elements are redundant. This test can be carried out in a very short time, but it is not without its drawbacks: the actual phoneme boundaries have to be known beforehand, and it is not easy to say how many missing bounds should be considered "too few". (Similarly, it is not known how many needless bounds should be judged as "too many".) Moreover, it is not clear when two phoneme boundaries can be mapped to each other.

Yet another possibility is simply to compute a mean squared error-type value. For each phoneme bound from the correct segmentation $T_I$, the closest bound from $T$ is taken, and their distance is raised to the second power. Then these values are summed up, and finally this sum is divided by the number of correct segmentation boundaries (i.e. $|T_I|$). The advantage of this measurement procedure over the previous one is that it seems more precise and more elegant from a mathematical point of view. Also, it rewards segmentation bounds placed closer to the correct ones, while the first method did not do so. But, unfortunately, its weaknesses are the same as before.

Since these testing methods have quite a few drawbacks, we eventually chose another option: we examined how a possible segmentation works as a part of a speech recognizer; that is, how it affects the actual recognition scores on real data. For this we simply performed exhaustive tests involving sentence recognition, and examined how the recognition scores (accuracy and correctness) varied; the higher the scores turned out to be, the better the segmentation was. A similar approach was employed in the SUMMIT System [70].

## 2.5 Our Speech Recognition Environment

What is left to define is the speech recognition environment where our tests were performed. As all our attempts were made in situations where we sought to improve something (either speed or accuracy), it might be of interest to ask under what conditions these improvements were made, whether a speech recognition framework was used, what feature set or database was used, and so on.

### 2.5.1 The OASIS Speech Laboratory

All experiments were performed in the framework of the OASIS Speech Laboratory [65]. This software system was developed by our Research Group on Artificial Intelligence. The system was designed with the aim of creating a general framework that is flexible enough to allow the experimentation with a wide range of techniques in speech recognition. The main strength of this system is its module-based structure, and its script-based execution, which makes experimental testing rather straightforward. Due to the module-based construction, another feature can be easily added to the whole system, such as new (machine-learning) features, some machine learning method (like Support Vector Machines (SVM) [90]), an application of some aggregation operator, or a search technique. Moreover, the script-based interface makes it easy to set the internal mechanisms of these modules via a number of parameters.

As there are many new ideas and techniques introduced throughout this work, the above proved to be very useful, as many new modules had to be added, and their behaviour had to be optimized via their parameters. Due to the script system, this parameter setting could even be done automatically (i.e. from another application), when some parameter had to be optimized.

### 2.5.2 The Feature Sets Used

Although none of the feature sets used were designed by the Author, they might be of relevance. Of course as the features are used for phoneme classification, which was done differently in a segment- and in a frame-based context, the feature sets used in them differed as well.

The main problem for feature sets in a segmental model is that for each phoneme an equal-sized feature vector has to be extracted, but as phoneme lengths may vary, these have to be calculated based on a varying number of observations. To overcome this problem usually average values are used on different parts of the segment to be categorized, or point-like values are used on specific locations (like at the beginning or at the end of the segment). In a frame-based model, of course, there is obviously no such difficulty as we classify exactly one frame at a time, making feature extraction more straightforward.

In the segment-based environment two feature sets were applied. The first one was constructed by Tóth [64; 101], and was based on Bark-band energies and their transformations after applying the non-linear adaptive gain control (AGC) algorithm [59]. The latter experiments were performed with the feature set of the SUMMIT System of MIT [45]. It is based on the 12 MFCC values: first these are averaged over the beginning, middle and end of the segment (which have a 3:4:3 length ratio), resulting in 36 features. Then the MFCC $\Delta$ values (i.e. the first derivatives) are calculated, and they are averaged over the 20 millisecond-long neighbourhood of the beginning and the end of the segment, adding another 24 feature values. Finally the length of the segment is included in the set, giving a total of 61 features.

In the frame-based case just the 12 MFCC values are calculated, and the total energy is added as the 13th. Then the derivatives and the derivatives of the derivatives (i.e. $\Delta$ and $\Delta\Delta$) of these scores are added, resulting in 39 features altogether [50].

### 2.5.3 The Databases Used

In this dissertation many experiments are described for the various methods used. Since these experiments were performed over a long period of time, larger and more relevant databases became available in the meantime, so several of them are used throughout this dissertation. But since many tests were performed on the same database, their detailed descriptions would be redundant in the experiments, so they are introduced here. Note that we consider the language model to also be part of a database, so these will now be described as well.

#### The Numbers Database

This was a rather small database consisting of utterances of Hungarian numbers. Basically the elements of such numbers were recorded (there are 26 of them), and each speaker uttered these elements twice, resulting in 52 utterances. The train database consisted of such recordings of $80$ people (men, women and children).

As for testing, this database only permitted that of word recognition. We created two test sets: the first consisted of these $26$ basic elements from six speakers, making a total of $312$ utterances. Test Set II contained more complicated numbers under 100, 169 test cases in all. Although this database seems to be quite small from the current point of view, it was good enough when the tests were made on it (around 2002). The segment-based baseline recognition scores were $97.44\%$ and $78.10\%$ ($304$ and $132$ correct hits) for Test Set I and Test Set II, respectively.

### The Children Database

The Children Database is a much more significant database. It is a corpus of 500 children uttering 60 words each, making a total of 30,000 utterances of 2,000 different Hungarian words with a variance related to everyday-use occurrence. From these utterances, 24,000 were used for training, and the remaining 6,000 served for testing purposes.

This database also did not include recorded sentences, thus only word recognition tests were permitted. Usually a set of 920 words were used for this purpose, the dictionary consisting of the original 2,000 words. To complicate matters, many of the children had just learned to read, which led to a diverse database, and many of the words were similar to each other with a phoneme difference of just one or two, and this problem is a difficult one for humans too. In such cases usually the context helps one to resolve it, but as the tests on this database were those of isolated word recognition, this helpful factor was unfortunately missing. As a consequence, the word error rate was usually around $20\%$ (which means word accuracy of about $80\%$). The *HTK* system [111] produced a word accuracy level of $84.34\%$. More details about this database can be found in [94].

### The Telephone Database (MTBA)

The Telephone Database was also a database of significant size. It consisted of 500 Hungarian speakers, each uttering 10 sentences and 4 words over the telephone. We used these utterances of 400 speakers for training purposes, and those of 100 speakers for various testing (like that of phoneme grouping, where these kinds of tests were carried out).

For word recognition we used another test database consisting of all these 500 speakers uttering the name of a different town or city. Some of these utterances were unrecognizable even to humans, 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 (town/city names). The *HTK* system [111] was used as a reference and produced a score of $92.11\%$ on this task, 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. More details about this database can be found in [107].

### The Medical Database

This last database was used for performing sentence recognition tests. In fact in this case the databases used for training and for testing had a different origin, but since we used this line-up in all tests incorporating sentence recognition, we can, from this point of view, treat them as one unit.

To train the phoneme models, we had a large, general database called MRBA: 332 people of various ages spoke 12 sentences and 12 words each, which were recorded on different computers and sound cards via different microphones [106]. After training, these phoneme classification models could be freely used to recognize utterances from any field, as long as the technical parameters of the recordings (sampling frequency, sampling rate) did not differ.

In the next step we combined this phoneme classification method with a simple language model. Here the sentences spoken were restricted to those of medical reports. The language model was a simple word 2-gram; i.e. the probability of the next word depended only of the last word spoken, and it was calculated by a statistical analysis of texts in a similar field. We carried out this investigation on all the available reports (nearly 9,000), which contained 2,500 different words in 95,000 sentences. (See [7] for more information.)

The combination of these two methods were then tested on 150 randomly selected sentences, which were naturally taken from the same type of texts, namely medical reports. (Otherwise the language model, which models sentences taken from this domain, would not have been of much use.) These sentences were recorded from 4 speakers, and they were tested one after the other. The basic recognition values were between $91\%$ and $98\%$, depending on whether we used segment- or frame-based features. These scores are probably due to the large number of words and the simple nature of the language model. The language model was refined later, but its inclusion would make the performances of some methods incomparable in this dissertation, so the experiments described here all used this "old" one.

# Chapter 3

# Reducing the Search Space in Speech Recognition

> *"Space is big. Really big. You just won't believe how vastly hugely mindboggingly big it is. I mean you may think it's a long way down the road to the chemist, but that's just peanuts to space."*
> *Douglas Adams – The Hitchhiker's Guide to Galaxy*

During the speech recognition process, a huge number of possible words have to be matched against the uttered speech signal. In this process many combinations of phoneme-sequences and the corresponding segments (*hypotheses*, in short) have to be handled. This large number of hypotheses leads to the need for an efficient search algorithm because, in the end, we only want the most probable among them; and because it is a well-known fact that the search process takes up most of the time requirements of speech recognition, this issue has been heavily investigated [56; 78]. How many unnecessary hypotheses are scanned before the best one is found depends only on the search method applied; thus, an "efficient" search method simply means that it is a fast one.

Another reason why we think that "good", in the case of search, just means "fast" is that, unlike most other subtasks, we cannot really improve the recognition accuracy of the speech recognition system by changing the search algorithm to a better one. After all, the task of the search process is to find the most *probable* finishing hypothesis from the set of *possible* hypotheses, while recognition accuracy depends on the relationship between the most probable hypothesis and the actual meaning of the utterance. If there is discrepancy between these two, it can be either because the correct word sequence is not allowed (so there is no *possible* hypothesis which includes it), or it is allowed, but it is not the most probable one. The former problem is influenced by the language modelling and segmentation subtasks, while the latter might be due to a fault of the signal processing part, or the phoneme classification subtask (or because the way we aggregate a hypothesis-level probability from the frame/phoneme-level ones is flawed), but the search method cannot be responsible for either of these.

This reasoning does not work backwards, however; by applying a search algorithm with a low efficiency, or setting its parameters badly, we can indeed reduce the recognition rate. It is so because most search methods are heuristics: they decide which hypotheses are worth extending, and which ones can be thrown away, to efficiently scan as small a part of the hypothesis space as possible. A badly configured search method, however, would discard many good hypotheses, and in the end would not result in the most probable word, but in some other one. So a good search method is not just fast, but it is fast *while preserving the recognition accuracy*.

In this section we will focus on the search problem in speech recognition. First the search space will be defined along with the search problem, then standard methods will be described for the search part. Next, we will introduce two methods for speeding up this search part of the speech recognition process. The first one constructs a multi-pass search method instead of the usual one-pass ones, based on a hierarchical grouping of phonemes; while in the second one we limit the possible search space (i.e. the set of possible hypotheses) by restricting the set of bounds between the phonemes that we have to choose from.

## 3.1 The Search Space

Next we will define the search space, the hypotheses and the way their scores are calculated. For it we will rely on the notation introduced in Section 2.2.1. Further, we apply the Unified View (also described in Section 2.2.1), which is the reason why these definitions more or less follow the segment-based approach.

The task of speech recognition is essentially a selection problem over a search space where the phonemes of the allowed words or word sequences (words in the following) are matched to the speech signal in a left-to-right manner. More precisely, the search space will be a Cartesian product space where the first dimension is a set of word prefixes, while the second is a set of segmentation-prefixes. Given a set of word sequences $W$, we use $Pref_k(W)$ to denote the $k$-long prefixes of all the phoneme sequences in $W$ (but only those words which have at least $k$ phonemes). Now let

$$I^k = \{[t_0, t_1, \ldots, t_k] : 1 = t_0 < t_1 < \cdots < t_k \leq t\} \tag{3.1}$$

be the set of sub-segmentations made of $k$ segments over the observation series $A = a_1 a_2 \ldots a_t$; i.e. they are prefixes of the possible segmentations consisting of exactly $k$ elements. (Thus all restrictions which were placed on a segmentation $I$ apply for $I^k$ except one, so $t_i$ has to be an integer, $t_0 = 1$, $t_i < t_{i+1}$, but $t_k$ does not necessarily equal $t$.) The hypotheses will be object pairs, i.e. they will be elements of

$$\mathcal{H} = \bigcup_{k=0}^{\infty} \left(Pref_k(W) \times I^k\right). \tag{3.2}$$

Practically speaking, a hypothesis is a beginning of a word and segmentation pair, where just the first $k$ phonemes (i.e. $o_1 o_2 \ldots o_k$) were positioned. These hypotheses can be arranged in a tree; we will denote the root of the tree – the initial hypothesis – by $h_0 = (\emptyset, [t_0])$ $(h_0 \in \mathcal{H})$, whereas $\mathit{Pref}_1(W) \times I^1$ will contain the first-level nodes. For a $(o_1 o_2 \ldots o_j, [t_0, t_1, t_2, \ldots, t_j])$ node we link all $(o_1 o_2 \ldots o_j o_{j+1}, [t_0, t_1, t_2, \ldots, t_j, t_{j+1}]) \in \mathit{Pref}_{j+1}(W) \times I^{j+1}$ nodes as its children. If we say that a hypothesis is extended, it means that we generate all hypotheses which are linked below it in this tree; that is, we add any valid single phonemes to the end of its phoneme sequence as $o_{j+1}$, and add every valid phoneme boundary to the end of its boundary set as $t_{j+1}$. ($t'$ is a valid phoneme boundary if $t_j < t'$ and $t' \leq t$. It is possible to set an upper limit on the length of a phoneme ($t_{max}$); then $t'$ is a valid phoneme boundary only if $t' - t_j \leq t_{max}$.)

In Chapter 2 we described the probability calculation process for whole words covering the whole utterance; of course it is easy to extend this definition so that it applies to hypotheses as well, i.e. for the nodes of this search tree. To this end let $g_1$ be a function which assigns a probability to a phoneme-speech segment pair, and let the $g_2$ function be an aggregation operator of some kind. Then, for a node $(o_1 o_2 \ldots o_j, [t_0, \ldots, t_j])$, its probability value is defined by

$$g_2\big(g_1([t_0, t_1], o_1), \ldots, g_1([t_{j-1}, t_j], o_j)\big). \tag{3.3}$$

Note that, in practice, it is worth calculating Eq. $(3.3)$ recursively. After defining the evaluation methodology we will look for a leaf with the highest probability. Obviously not every leaf will be suitable, just the ones which have a phoneme-sequence forming a correct word and not just a prefix of it (i.e. $(o_1 o_2 \ldots o_j) \in W$), and ending at the end of the speech signal ($t_j = t$). We will call the hypotheses which fulfil these requirements *finishing hypotheses*.

This definition in typical circumstances leads to a huge hypothesis space, where a full search would be unfeasible because of the big run time and memory requirements. This leads one to use heuristics like the well-known Viterbi beam search [48], or our choice, the multi-stack decoding algorithm [3], which will be described in more detail later on.

Actually, few search methods rely directly on this tree; most of them just use the initial hypothesis $h_0$, the set of finishing hypotheses, and the methodology of extending a hypothesis (i.e. continuing it with another valid phoneme on a valid segment). Naturally they also need a probability value for a hypothesis, and expect that it should not increase when extending some hypothesis. These methods can, of course, be easily derived from the search tree defined above.

Lastly we should mention that in the search process there usually arises the problem of underflowing. It means that a value close to $0$, because of the special characteristics of computer number representation, in practice becomes $0$. In our case it would happen quite frequently and it would result in a case where practically *all* considered hypotheses would have the same probability of $0$. To avoid this, a standard technique is normally employed: instead of a probability $p$, we use the cost $c = -\log p$. This will, of course, require other modifications like using addition instead of multiplication, and looking for the hypothesis with the lowest cost, and not the one with the highest probability.

## 3.2 Common Search Techniques

Having specified the search space, we can start looking for the best word-segmentation pair in it. In theory we may use any general search method for it (e.g. a depth-first or breadth-first search), but in practice most methods would prove unfeasible due to their time and memory requirements. Fortunately, in the literature there exist many search techniques especially designed for our particular problem. In the following we will list the more important ones.

### 3.2.1 Time-synchronous Search Algorithm

There might be situations (although quite rarely), where we need the optimal solution in every case. Even then, besides examining *every* possible hypothesis (which is highly impractical due to the computational time required), we still have one choice left: we can apply a time-synchronous decoding method using dynamic programming [79]. This algorithm can be used both in a frame-based and in a segment-based context.

This method employs a table with the possible bounds between the phonemes (so the possible $t_j$ values) labelling the columns and the possible phoneme-sequences labelling the rows. Each cell contains the lowest cost of the hypotheses that have the corresponding phoneme-sequence and ending at the corresponding time instance. To determine the value of a cell we take the value in a cell belonging to an earlier time value and to the phoneme-sequence without its last phoneme, and add up the cost of this last phoneme on the interleaving speech segment. (So, technically, we expand the earlier hypothesis with the last phoneme in such a way that its child has to end at the actual time instance.) The value of the cell will be the minimum of these sums. (For the pseudocode of the algorithm, see Appendix A.5.) A drawback of this method is, however, that we need to fill a separate table for each word in the dictionary, although various tricks – such as reusing the table-parts of the words with the same prefix – can also be applied. A similar algorithm used in the case of a Hidden Markov Model is known as the *Viterbi algorithm*.

In practice the full Time-synchronous Search Algorithm is rarely used, mainly for two reasons. The first is that to employ it, we need to test every possible phoneme sequence, which could be a very large number if we do not limit our investigations to simply words. (There can indeed be many sentences, even if they are constructed from just a small number of words.) The second one is that we rarely need the most probable hypothesis, and an estimate of it (which will probably differ only in one or two segmentation bounds by a few milliseconds earlier or later) is definitely not enough; and for the other cases there exist search heuristics which are much faster than this algorithm. Its use can probably be limited to automatic segmentation (forced alignment), where the correct phoneme sequence is already given, a small difference in the segmentation can be important, and there is no need for a real-time run. Otherwise, in practical situations the Time-Synchronous Search Algorithm is usually applied with a beam search heuristic (see Section 3.2.5).

### 3.2.2 Stack Decoding Algorithm

The remaining methods are all heuristics; so they do not necessarily return with the optimal finishing hypothesis, but this drawback is usually compensated by the fact that they run much faster. And fortunately, in typical recognition circumstances, an almost-optimal solution is good enough.

There is another common aspect of the following algorithms: they all employ a data structure called a *stack* (not to be confused with the LIFO and FIFO type stacks, which are essentially different data structures). A stack is basically a list which stores hypotheses along with their cost, and has the basic operation of insertion. Also, it provides easy access to the hypothesis with the lowest cost, and the removal of this hypothesis is supported too. (The combination of these two operations, as in the LIFO stacks case, is called *popping*.) Some algorithms also employ limited-size stacks; this means that just the best $n$ hypotheses are stored at any time, so if an $n+1$th hypothesis is inserted, the one with the highest cost will be thrown away. (We may say that this hypothesis was *pruned*.) In our tests, stacks implemented by storing the hypotheses and their costs in a red-black tree [17] were used.

The stack decoding algorithm employs just one stack, so every hypothesis to be considered later is stored there. As a result, it is time-asynchronous, which means that, unlike the Time-synchronous Search Algorithm or the Viterbi beam search, it compares hypotheses with different ending times. In the first step we place the initial hypothesis $h_0$ into the stack. Then we pop this hypothesis in order to examine and extend it. Next, we put all these newly generated hypotheses into the stack and pop the one which has the lowest cost of them ("best-first search"). We repeat the process until the popped hypothesis reaches the end of the utterance (and it is a finishing one) [4; 52; 110]. For the pseudocode of this algorithm, see Appendix A.1.

The reason why this algorithm works is that it extends hypotheses, and their cost increases since we add these costs (the cost of the original hypothesis and the cost of the new phoneme on the new speech segment, which are non-negative real numbers) together. Thus, when we pop a hypothesis which has reached the end of the utterance, all unexamined hypotheses in the stack will have higher costs than our actual solution; moreover, their extensions would have even higher ones, so they do not have to be examined any further.

If the algorithm were what we have described so far, it would supply the optimal solution. But in practice it is inevitable to use a finite-sized stack, which explains why we called the stack decoding method a heuristic. For large vocabularies and/or sentences (i.e., those with a huge search space) there is a risk that it will eliminate the best scoring hypotheses having a later end time, because many much shorter hypotheses will have lower costs. This behaviour can be controlled by the *stack size* parameter: if we store fewer hypotheses at a time, we will probably finish faster, but the risk of throwing away the correct one becomes more likely; on the other hand, a larger stack size means greater accuracy, but also a higher running time. There is another weak point of this method: by increasing the length of an utterance, the run time of the stack decoding algorithm will increase exponentially, for the same reason.

### 3.2.3 An A* heuristic

The A* search algorithm [86] is also a common method for finding a near-optimal solution in various fields of Artificial Intelligence, and it can also be applied in speech recognition. In it, besides the cost of the actual hypothesis – which will be denoted by $g(H)$ –, there is a $h(H)$ value for estimating the cost of the remaining path up to the end of the utterance. We put the hypotheses into a stack and sort them using the value $f(H) = g(H) + h(H)$. Essentially, this is just a variation of the *stack decoding* method as it only differs from it in this respect. (For the pseudocode of this algorithm, see Appendix A.2.)

Jelinek offers a method for constructing a heuristic based on examples [53]. 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, which should then give a good estimate of the cost for the remaining part. 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 it is fairly straightforward to satisfy with the latter. However, when we calculate the minimum cost per unit time using the latter version, there is a probable loss of efficiency although it should still be somewhat better than the simple stack decoding method [50].

### 3.2.4 Multi-stack Decoding Algorithm

In the multi-stack decoding algorithm we use several stacks: a separate one is assigned to each possible time instance, and we store the hypotheses in the stack corresponding to their end times [3; 73]. In the first step we place the initial hypothesis $h_0$ 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. The result will be the first finishing hypothesis which is popped from the stack assigned to the final time instance (i.e. the most probable one with a phoneme sequence belonging to a correct word). Appendix A.3 shows the pseudocode of the multi-stack decoding method.

All the stacks used are of a limited size: if there are too many hypotheses in a stack, we prune the ones with the highest costs. So this algorithm has one parameter, the *stack size*, which regulates its behaviour: 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. Above a certain parameter value there is no change in accuracy and this is often what we call the optimal value.

Note that this algorithm was chosen as our basic search technique throughout this dissertation so, unless stated otherwise, the multi-stack search method is the one we shall employ.

### 3.2.5 Viterbi Beam Search Algorithm

This algorithm can be regarded as a variation of the multi-stack decoding method, which differs in only one aspect: instead of keeping the $n$ best hypotheses, a variable $T$ called the *beam width* is employed. For each time instance $t_i$ we calculate $C_{\min}$, i.e. the lowest cost of the hypotheses with the end time $t_i$, and prune all hypotheses from the appropriate stack whose cost $C$ is larger than $C_{\min} + T$ [48; 50; 69]. For the pseudocode of the algorithm, see Appendix A.4.

## 3.3 Ways of Improving the Efficiency of a Search

As we discussed earlier, the search problem is a very important issue because it affects the speed of the speech recognition process quite a lot. Also, if we choose the search method incautiously, or we do not configure it properly (if the method can be tuned), we can also ruin the accuracy of the system by forcing it to discard good hypotheses which do not appear so promising at that particular time instant. Thus it is an area worth investigating, because there can be no search technique which appears to be "too fast".

In this section we will introduce two methods which make this search procedure faster. The first is a search technique which breaks down the search into more steps, where the next step is always a more accurate (and thus, slower) examination of fewer and fewer hypotheses. The second one, however, limits the choice of applicable boundary positions between phonemes. What these two methods have in common is that they simply apply a standard search algorithm, but the hypotheses that this algorithm has to examine are restricted in some way.

### 3.3.1 Hierarchical Phoneme Classification

Next we will define a novel search technique. We use the basic and well-known idea of a multi-pass search [50; 92]; it means that we divide the search part into more steps, each one further restricting the possible set of hypotheses. The idea behind this trick is that many of the hypotheses have a very high cost (i.e. they are very improbable), thus they can be excluded via some quick checks. This idea is used in various fields of Artificial Intelligence [33; 104], but it is not uncommon in speech recognition either [80; 81].

To create these quick check passes, we construct a hierarchy of the available phoneme set by automatic means, hence the earlier steps use more compact phoneme groups. However this construction of the phoneme groups is not trivial, so the choice of the algorithm we use strongly affects the speed and recognition accuracy of the speech recognition system.

#### Multi-pass Search Strategies

In general, **multi-pass methods** work in two or more steps: in the first pass the less likely hypotheses are discarded based on some condition requiring low computational time. Then, in the later passes, just the remaining hypotheses are examined by more complex, reliable evaluations, which will approximate the probabilities of the hypotheses more precisely. In the last pass, of course, we have to make a search with the original thoroughness; but, hopefully, the earlier steps will have reduced the set of possible hypotheses to a level which makes it possible to perform this last step quite fast.

In a conventional two-pass procedure only one such quick check is done, then all the remaining hypotheses are examined in the final step [15]. 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 [2; 27].

To speed up the earlier steps, we need to construct faster phoneme classifiers, and the most straightforward way of doing this is to reduce the number of features. Fewer features means a quicker (though not as accurate) phoneme classification, which leads to a faster recognition process. For example in our case, where Artificial Neural Networks are used, it leads to a smaller number of *input* neurons, and the hidden layer(s) can be reduced as well.

We, however, decided to do it the other way around: the number of phoneme groups (i.e. the number of *output* neurons of the phoneme-identifying ANNs) was decreased by fusing different, but similar phonemes into one group. 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 only the successful words of the previous level. Obviously, during the last pass we had to use the original phoneme set to get the final recognition result. In this pass, unlike the former ones, we were only interested in the best hypothesis (in contrast with the earlier passes, where we examined a set of words). At each level we employed the conventional multi-stack decoding algorithm in the search process.

### Clustering the Phoneme Set

Now we shall explain the technique we introduced to create smaller, more compact phoneme groups by clustering. First we will define two novel similarity functions between phonemes, demonstrate that they have the right sort of properties to be distance functions, then utilize them in the phoneme-clustering problem.

Of course, combining phonemes into higher units is not a completely new idea. Still, our approach described here has a number of novel properties. First, it performs this fusing to create a multi-pass search. Second, this combination could be based on a priori, external phonetic knowledge, as this area has been extensively investigated by linguists [5]; instead, here it is constructed automatically, based on the properties of the phoneme classifier. This way it automatically takes into account its environment, which should lead to better performance than the application of an external knowledge. And finally, for a multi-pass search like this we need the hierarchical property of the phoneme groups used. In our algorithm it arises naturally, whereas in the other case we have to ensure that it is present.

To explain the process, first some terms have to be defined. The task of a classifier is to classify a set of observations (or features) into one of the previously set $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ classes. The behaviour of a classifier can be best described by the widely-used *confusion matrix* $M$: it is constructed such that $m_{i,j}$ is the number of examples (in our case, actual phonemes) belonging to $\omega_j$ from a selected test set which were recognized as $\omega_i$s by the classifier [26; 66]. (For an example, see Table 3.1.) In our case the classifier is used to categorize some segments of speech into one of the phoneme classes. The confusion matrix of a good classifier is close to a diagonal matrix, so we only concentrate on the number of misclassified items (i.e. the number of examples that appear outside the diagonal of the confusion matrix).

Grouping phonemes is a standard *clustering problem* [46]: some points (in our case the phoneme classes) are to be assigned to a certain number of *clusters* (in our case to phoneme groups). There are quite a few general algorithms for this task. We are going to use the Agglomerative Hierarchical Clustering algorithm [46], which needs a distance function for two clusters. This distance function will be defined below, but first we will explain how this algorithm works, and in this step we shall assume that the distance function (denoted by $\mathcal{D}(C_i, C_j)$ for the clusters $C_i$ and $C_j$) is already given.

At the start each phoneme will be regarded as a cluster by itself. Then, in each step, we locate the $C_i$ and $C_j$ clusters where $\mathcal{D}(C_i, C_j)$ is minimal, and fuse them: their elements (in our case, the original phonemes) are all put into one cluster. We repeat this until the minimal distance reaches a given parameter $L$, i.e. $\forall i \forall j \; i \neq j \; \mathcal{D}(C_i, C_j) \geq L$. (See Appendix A.6 for the pseudocode of this algorithm.)

Now we define our distance functions $\mathcal{D}(C_i, C_j)$ used in clustering. First we define the normalized matrix $M'$ for the confusion matrix $M$ of the applied phoneme classifier; it takes the form

$$m'_{i,j} = \frac{m_{i,j}}{\sum_k m_{k,j}}, \quad 1 \leq i, j \leq K \tag{3.4}$$

where $K$ is the number of classes. We may assume that $\sum_k m_{k,j} \neq 0$, otherwise it would mean that the $j$th phoneme has no examples at all. (In this case this phoneme should first be manually grouped with another existing phoneme, or it should be deleted from the list of phonemes.) For an example of a normalized confusion matrix like this, see Table 3.2. Now the distance function can be defined based on this $M'$ matrix. For it, we introduced two variations:

$$d^1_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } m'_{i,j} = m'_{j,i} = 0 \text{ and } i \neq j \\ -\log(m'_{i,j}) & \text{if } m'_{j,i} = 0 \text{ and } m'_{i,j} \neq 0 \\ -\log(m'_{j,i}) & \text{if } m'_{i,j} = 0 \text{ and } m'_{j,i} \neq 0 \\ \min\big(-\log(m'_{i,j}), -\log(m'_{j,i})\big) & \text{otherwise,} \end{cases} \tag{3.5}$$

and

$$d^2_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } m'_{i,j} = m'_{j,i} = 0 \text{ and } i \neq j \\ -\log\big((m'_{i,j} + m'_{j,i})/2\big) & \text{otherwise.} \end{cases} \tag{3.6}$$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2502 | 3 | 96 | 35 | 4 | 0 | 0 | 4 | 8 | 0 | 12 |
| 2 | 18 | 965 | 3 | 24 | 3 | 0 | 0 | 0 | 5 | 1 | 49 |
| 3 | 87 | 8 | 875 | 19 | 2 | 0 | 0 | 5 | 11 | 0 | 18 |
| 4 | 43 | 11 | 16 | 271 | 1 | 1 | 0 | 1 | 2 | 0 | 12 |
| 5 | 12 | 2 | 3 | 2 | 2250 | 257 | 80 | 101 | 53 | 5 | 48 |
| 6 | 0 | 1 | 0 | 0 | 51 | 299 | 17 | 22 | 8 | 24 | 17 |
| 7 | 1 | 0 | 0 | 0 | 46 | 31 | 208 | 6 | 1 | 15 | 5 |
| 8 | 3 | 4 | 3 | 1 | 70 | 39 | 8 | 5235 | 111 | 19 | 116 |
| 9 | 7 | 1 | 6 | 3 | 19 | 10 | 2 | 97 | 461 | 2 | 77 |
| 10 | 1 | 0 | 0 | 0 | 12 | 88 | 25 | 62 | 11 | 830 | 8 |
| 11 | 39 | 71 | 21 | 31 | 38 | 23 | 19 | 102 | 367 | 18 | 2316 |

Table 3.1: An example of a confusion matrix coming from a phoneme classification problem

So in a typical case $d^1_{i,j}$ will determine the distance between the two phonemes by considering the higher ratio of misclassified items, while $d^2_{i,j}$ will use an average value. Next let $D'$ be the output of some shortest path-finding algorithm (e.g. Dijkstra's algorithm [20]) with the input of either the $D^1$ or $D^2$ matrix; this $D'$ will contain the distance between the phonemes. Since it cannot be known in advance which distance function will turn out to be better in practice, we will deal with both of them from now on (and simply refer to both of them by $D'$), and we will also test both variations. It can readily be seen that $D'$ is a distance function, moreover it satisfies the criteria of being a metric because

- $d'_{i,i} = 0$,
- $d'_{i,j} = d'_{j,i}$, and
- $d'_{i,j} \leq d'_{i,k} + d'_{k,j}$.

Now we have to define the distance $\mathcal{D}(C_i, C_j)$ between the clusters $C_i$ and $C_j$ from the $d'(x_i, y_i)$ values (the distance between different phonemes). To do this we also have two straightforward options [46]:

$$\mathcal{D}_{\min}(C_i, C_j) = \min_{x,y} \{d'(x, y) | x \in C_i, y \in C_j\}, \tag{3.7}$$

which is called *complete-linkage clustering*, and

$$\mathcal{D}_{\max}(C_i, C_j) = \max_{x,y} \{d'(x, y) | x \in C_i, y \in C_j\}, \tag{3.8}$$

which is called *single-linkage clustering*. The former tends to create longer, larger clusters, while the latter usually creates more compact ones. We tested both versions.

We should mention here that the use of $\mathcal{D}_{\max}$ in the clustering algorithm could lead to a nondeterministic case if, at any given point, there exist some clusters $C_i$, $C_j$ and $C_k$ such that $\mathcal{D}_{\max}(C_i, C_j) = \mathcal{D}_{\max}(C_i, C_k)$. On the other hand $\mathcal{D}_{\min}$ is not a metric

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 922 | 3 | 94 | 91 | 2 | 0 | 0 | 1 | 8 | 0 | 4 |
| 2 | 6 | 905 | 3 | 62 | 1 | 0 | 0 | 0 | 5 | 1 | 18 |
| 3 | 32 | 8 | 855 | 49 | 1 | 0 | 0 | 1 | 11 | 0 | 7 |
| 4 | 15 | 10 | 16 | 702 | 0 | 1 | 0 | 0 | 2 | 0 | 4 |
| 5 | 4 | 2 | 3 | 5 | 901 | 344 | 223 | 18 | 51 | 5 | 18 |
| 6 | 0 | 1 | 0 | 0 | 20 | 400 | 47 | 4 | 8 | 26 | 6 |
| 7 | 0 | 0 | 0 | 0 | 18 | 41 | 579 | 1 | 1 | 16 | 2 |
| 8 | 1 | 4 | 3 | 3 | 28 | 52 | 22 | 929 | 107 | 21 | 43 |
| 9 | 2 | 1 | 6 | 5 | 8 | 13 | 6 | 17 | 444 | 2 | 29 |
| 10 | 0 | 0 | 0 | 0 | 5 | 118 | 70 | 11 | 11 | 908 | 3 |
| 11 | 14 | 67 | 21 | 80 | 15 | 31 | 53 | 18 | 354 | 20 | 865 |

Table 3.2: An example of a normalized confusion matrix (the values have been multiplied by 1000 for the sake of clarity). The columns should be add up to about 1000 (rounding is responsible for the inexact values)

because in some cases the triangle inequality does not hold: there exist $C_i$, $C_j$ and $C_k$ clusters such that $\mathcal{D}_{\min}(C_i, C_j) \nleq \mathcal{D}_{\min}(C_i, C_k) + \mathcal{D}_{\min}(C_k, C_j)$.

At this point, having chosen the clustering method, and having defined the starting clusters and the distance functions used, the phoneme grouping method is almost ready for use. All that is left is to set the stopping criterion, i.e. the value (or values) for $L$. For this purpose we suggest a more thorough evaluation on the test set. That is, applying the clustering algorithm will lead to a series of unions and a series of the corresponding distance values. We will choose the possible values of the limit $L$ based on them, which will result in phoneme groups that will be used in the recognition process. We suggest those values for $L$ where there is a significant gap between successive distance values in the output, as these mean that the merging of significantly distinct clusters than before will occur. Figure 3.1 and Figure 3.2 show graphs with the number of phoneme groups depending on the value of $L$ for a given test set. In a typical figure the longer horizontal lines indicate good values for $L$.

Also note that for a given speech recognition test we may use several earlier passes (by choosing multiple $L$ values). In such a case the phoneme groups of a latter pass can be derived from the earlier one by dividing some phoneme sets into more groups, which is why we called this method a hierarchical phoneme clustering method. This property will definitely come in handy when performing the actual multi-pass speech recognition process.

### Tests

At this point our proposed multi-pass search technique is ready for use; but to measure its performance, we have to define the speed of a multi-pass search configuration. We will measure the running time of a search method by the number of phoneme-identifications, which is proportional to the actual running time, but it is easier to measure; then we will calculate the amount of speed-up achieved by taking the inverse

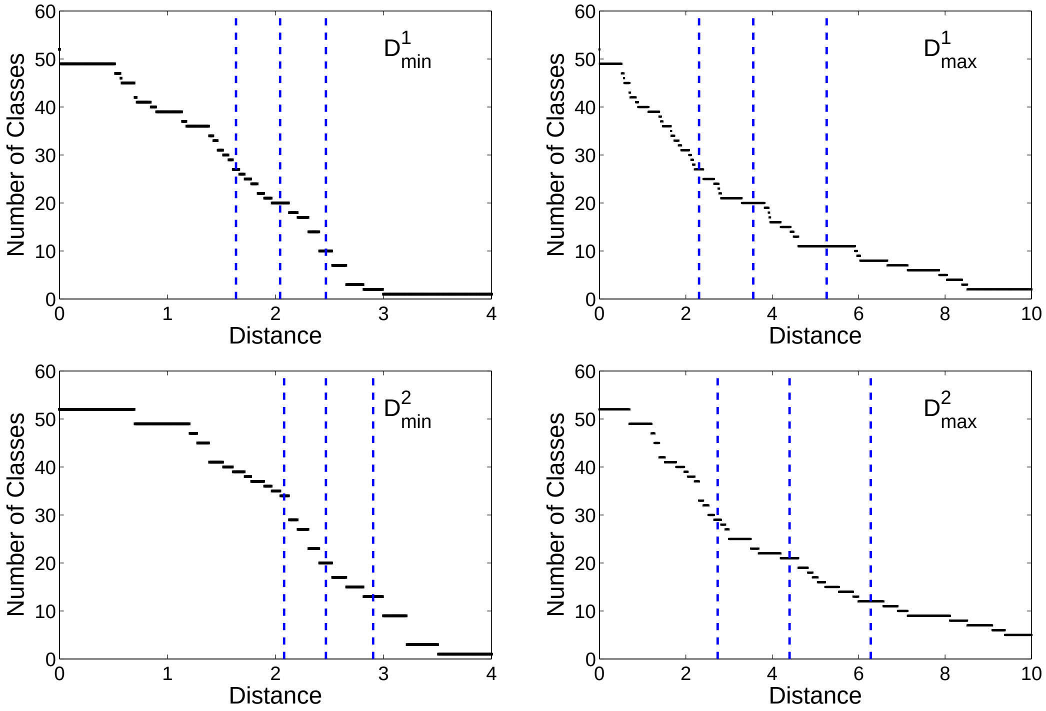

Figure 3.1: The number of phoneme groups (classes) – $L$ limit diagram for the four distance-variations on the Children Database (isolated word recognition, segment-based context), and the values chosen for the appropriate passes.

of these values. The problem is that as we use several different phoneme classifying ANNs at the same time, their size plays a significant role in the speed-up process. Fortunately though, all of these are traditional feed-forward ANN-s with one hidden layer, and the ratio of evaluation times for such ANNs can be easily calculated from the number of input, output and hidden neurons, so it was rather straightforward to calculate how much faster the evaluation of a smaller neural network is than the original one. Afterwards, the results for all the passes were summed up to normalize the running time of the whole multi-pass recognition process to that of the final pass using the original phoneme set (and hence it used the original ANNs).

At each pass the multi-stack decoding algorithm was applied, altering the stack size parameter for each pass separately, which was necessary to make the sum of all passes faster than the original one. For each tested multi-pass configuration we tested each stack size combination from a given set of values, and chose the one which satisfied the minimal rate of accuracy set and was the fastest among them.

### Results for the Children Database

This multi-pass method was tested on two different databases. The first one was the Children Database (see Section 2.5.3), in an isolated word recognition task. The model we applied was a segment-based one using the feature set introduced by Tóth [64], with a phoneme recognition rate of $80.57\%$ on the original phoneme set, and it remained

| | $d^1$ | | $d^2$ | |
|---|---|---|---|---|
| Passes | $\mathcal{D}_{\mathrm{min}}$ | $\mathcal{D}_{\mathrm{max}}$ | $\mathcal{D}_{\mathrm{min}}$ | $\mathcal{D}_{\mathrm{max}}$ |
| $\mathcal{P}_1$ | 28 | 27 | 34 | 29 |
| $\mathcal{P}_2$ | 17 | 20 | 20 | 21 |
| $\mathcal{P}_3$ | 9 | 11 | 13 | 12 |

Table 3.3: Number of phoneme groups for the various distance functions and passes on the Children Database (isolated word recognition, segment-based context). The original phoneme set ($\mathcal{P}_0$) consisted of 52 phonemes.

| Used passes | | | | $d^1$ (minimum) | | $d^2$ (average) | |
|---|---|---|---|---|---|---|---|
| $\mathcal{P}_0$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | $\mathcal{D}_{\mathrm{min}}$ | $\mathcal{D}_{\mathrm{max}}$ | $\mathcal{D}_{\mathrm{min}}$ | $\mathcal{D}_{\mathrm{max}}$ |
| • | ○ | ○ | ○ | 100.00% | 100.00% | 100.00% | 100.00% |
| • | • | ○ | ○ | 187.81% | 188.96% | 270.96% | 233.32% |
| • | ○ | • | ○ | 217.46% | 188.23% | – | 160.61% |
| • | ○ | ○ | • | – | – | – | **338.70%** |
| • | • | • | ○ | 162.76% | 229.64% | – | 172.62% |
| • | • | ○ | • | – | – | – | 189.22% |
| • | ○ | • | • | – | – | – | 248.47% |
| • | • | • | • | – | – | – | 194.60% |

Table 3.4: Speed-up results for our multi-pass method for the Children Database (isolated word recognition, segment-based context). The • symbol means we applied the given pass in the given configuration, while the ○ symbol means that we omitted it. The percentage values show the speed of the fastest configuration using the given passes, as the speed-up ratio achieved over the original one-pass search.

around $80\%$ when we applied the restricted phoneme sets. The diverse nature of the database led to a basic word recognition percentage of $84.14\%$ with the OASIS Speech Laboratory, whereas the *HTK* system [111] we used as a reference achieved a score of $84.34\%$. In our tests we expected a word-level accuracy of at least $80\%$ for a multi-pass configuration. The results of this experiment were published in [39].

In Figure 3.1 we plotted the number of phoneme groups as a function of the limit $L$ and identified the bigger flat regions in each curve as they represent promising stopping values. For each of them we selected three values for $L$, resulting in the same number of phoneme groups used in the multi-pass recognition method. (Note that some promising flat regions like the region around 3.5 for $\mathcal{D}^2_{\mathrm{max}}$ had to be ignored to avoid passes with almost the same number of phonemes.) The corresponding recognition steps were called Pass 1 ($\mathcal{P}_1$), Pass 2 ($\mathcal{P}_2$) and Pass 3 ($\mathcal{P}_3$), with the number of phoneme groups varying from 27 to 34, from 17 to 21 and from 10 to 13, respectively (see Table 3.3). The default phoneme set was labelled $\mathcal{P}_0$ and had 52 phonemes.

We had four possible ways of defining the distance function and thus constructing these phoneme sets, and all four were tested. In Table 3.4 the performance of all four are shown along with the amount of speed-up achieved by them, which is linearly proportional to the reciprocal rate of running speeds. "•" means that we applied the

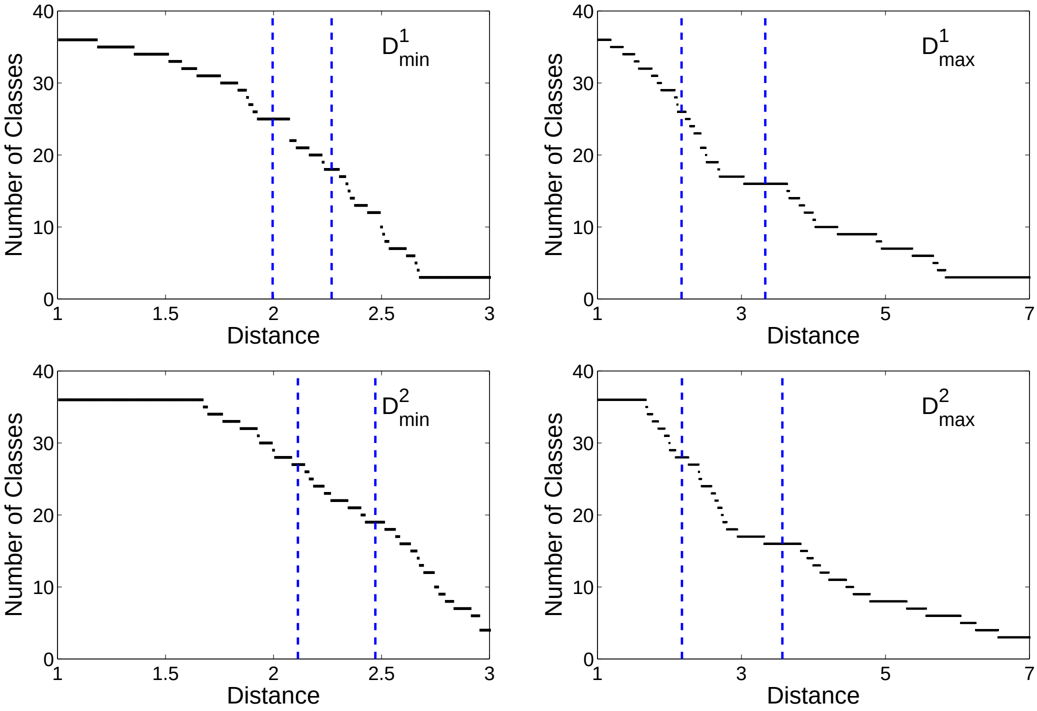

Figure 3.2: The number of phoneme groups (classes) – $L$ limit diagram for the four distance-variations on the Telephone Database (isolated word recognition, frame-based context), and the values chosen for the appropriate passes.

given recognition pass in the configuration, while "○" indicates that this pass was omitted; "–" means that the given configuration could not attain the required minimal recognition accuracy; $d^1$ and $d^2$ denote the chosen distance function between phonemes, while $\mathcal{D}_{\min}$ and $\mathcal{D}_{\max}$ denote the chosen distance function between phoneme groups.

Examining the results led us to the following observations. First, it was indeed possible to speed up a speech recognition system with a multi-pass search method by creating phoneme groups in the way we proposed. Because the last pass always applies the original phoneme set and thus with the original phoneme classifier, a faster multi-pass search algorithm means that in the earlier passes the list of possible words was drastically reduced, thus in the last pass it was enough to use small-sized stacks. Second, it seems that using $\mathcal{D}_{\min}$ in the clustering algorithm usually leads to a worse result than $\mathcal{D}_{\max}$, which suggests that the compactness of a phoneme group is very important. Third, we noticed that two-pass searches performed better than three- or four-pass configurations. The one-pass configuration was the slowest of the ones we tested among those that were to able to achieve an accuracy of at least $80\%$.

### Results for the Telephone Database

The other database we used for testing was the Telephone Database [107] (for details, see Section 2.5.3). We used the utterances of the 100 individual speakers for the phoneme-construction task, then performed an isolated word recognition on the 431

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

Table 3.5: Number of phoneme groups for the various distance functions and passes on the Telephone Database (isolated word recognition, frame-based context). The original phoneme set ($\mathcal{P}_0$) consisted of 36 phonemes.

| Used passes |  |  | $d^1$ (minimum) |  | $d^2$ (average) |  |
|---|---|---|---|---|---|---|
| $\mathcal{P}_0$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{D}_{\min}$ | $\mathcal{D}_{\max}$ | $\mathcal{D}_{\min}$ | $\mathcal{D}_{\max}$ |
| • | ○ | ○ | 100.00% | 100.00% | 100.00% | 100.00% |
| • | • | ○ | 174.82% | **241.20%** | 204.61% | 203.09% |
| • | ○ | • | 160.95% | 216.21% | 151.91% | **239.59%** |
| • | • | • | 113.29% | 184.68% | 126.18% | 168.30% |

Table 3.6: Speed-up results for our multi-pass method for the Telephone Database (isolated word recognition, frame-based context). The • symbol means we applied the given pass in the given configuration, while the ○ symbol means that we omitted it. The percentage values show the speed of the fastest configuration using the given passes, as the speed-up ratio achieved over the original one-pass search.

city names. The *HTK* system [111] used as reference yielded a score of $92.11\%$ here. Our model was a frame-based one, with the standard 39 MFCC $+ \Delta + \Delta\Delta$ coefficients as features for ANNs. The baseline recognition accuracy of this system was $95.12\%$, whose value (and the speed of the corresponding recognition configuration) served as our baseline. The results of this experiment were published in [63].

We constructed the new phoneme groups in a similar way to those for the Children Database: we tested all four clustering configurations, and plotted them in Figure 3.2 as a function of $L$ where we looked for the bigger flat regions. On each curve we chose two $L$ values, resulting in two phoneme groups, which were then 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 (see Table 3.5). The default phoneme set had 36 phonemes and the step belonging to it was denoted by $\mathcal{P}_0$. The lower number of original phonemes (compared to the previous test) was due to the fact that this test was a frame-based one, thus no distinction was made between long and short phoneme pairs; this smaller initial phoneme set then led to the lower number of possible earlier passes. 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 to 200 with a step size of 25. For example, for a three-pass configuration it meant 512 test cases. In the end for each combination we kept those stack sizes that produced the highest speed-up and had a word accuracy score of at least $94\%$.

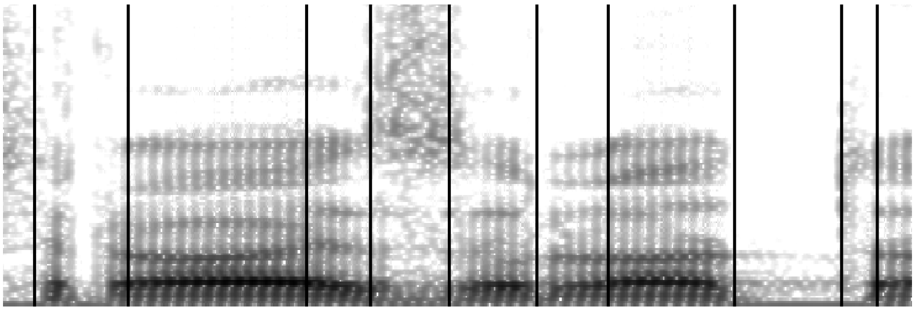

Figure 3.3: The spectrum of a speech excerpt, along with its correct segmentation.

Examining the results in Table 3.6, we can confidently say that the search method presented proved to be successful on this database too: while maintaining this level of recognition accuracy, a multi-pass recognizer ran 13–143% faster than the basic multi-stack decoding method (meaning a 12–59% reduction of the actual running times). It is also true that two-pass methods in general were faster than three-pass ones, and among them, the two-pass methods with $\mathcal{P}_1$ were usually better than the ones with $\mathcal{P}_2$. This suggests that there is no need to reduce the number of phoneme groups below 25. Since the initial phoneme set only consisted of 36 phonemes, there was no point in creating more phoneme groups; it is possible to perform searches with four or more passes, but a larger original phoneme set would be needed to justify this, as in the case for the Children Database.

We can also 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 $\mathcal{D}_{\max}$, which is the same result as we found on the Children Database. On the other hand, the performance of $d^1$ and of $d^2$ seem to be quite similar, thus we cannot tell which one is better.

### 3.3.2 Speeding Up Segment-Based Speech Recognition via Segmentation

In this part we will examine methods for another area in speech recognition to make speed-ups. We will turn to the segmentation subtask of the speech recognition process, so we will restrict the set of possible boundary positions between phonemes. This could cause a significant speed-up because the set of possible hypotheses becomes heavily restricted. For example, if we use the multi-stack decoding algorithm, then a separate, equal-sized stack is needed for each possible phoneme boundary; so, with this restriction, a proportional (or even greater) speed-up can be attained.

It is common to apply some sort of segmentation in a segment-based environment, but usually just a quite dull, equidistant solution is performed. (Most frame-based systems effectively treat every frame a possible segment bound.) If we can make the available phoneme boundaries fall more closely at the transition of the uttered phonemes, we should even be able to raise the recognition scores a bit.

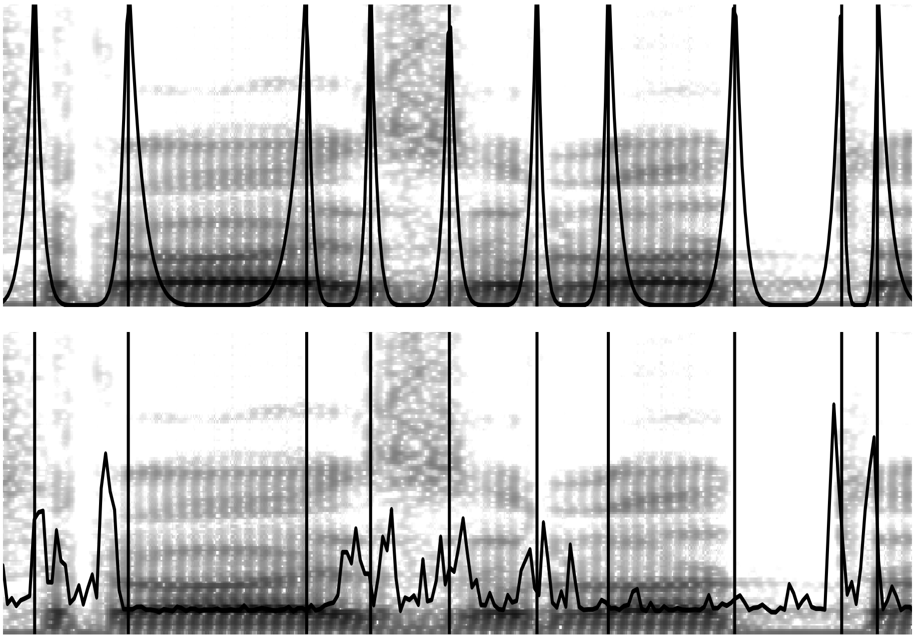

Figure 3.4: Up: the same spectrum as in Figure 3.3 with the correct segmentation and the curve used as the training target for the ANNs. Down: the same spectrum with the correct segmentation and the resulting $b_i$ values.

### The Concept of Segmentation

In this part we will refer to the segments $A_1, \ldots, A_n$ with their boundary elements, i.e. $I = [t_0, t_1, \ldots, t_n]$ ($t_j < t_{j+1}$ and $1 \leq t_j \leq t$, $t$ being the length of the utterance). This $I$ is called a *segmentation*, and we can refer to an $A_j$ segment with its starting and ending segmentation bounds as $[t_{j-1}, t_j]$. Now we will limit the range of these $t_j$ values: instead of taking any integer between $1$ and $t$, only the elements of some set $T$ will be allowed. This set is called the set of possible segmentation bounds (or *possible segmentation* in short), while the algorithm which supplies these values for the given speech signal $A$ is called the *segmentation algorithm*.

It is not hard to justify why we should restrict the set of possible segmentation bounds to the elements of this set $T$. It is very unlikely that every value between $1$ and $t$ will be needed (partly because it would mean that all phonemes are then of identical length and more importantly because a typical length of a frame is much lower than that of a phoneme). On the other hand, keeping values in $T$ which will obviously not be used in any segmentation makes speech recognition much more time-consuming without any gain. Constructing a good $T$ also seems possible because in the ideal case it coincides with the real boundary between phonemes, which can hopefully be predicted via standard signal processing techniques.

### Related Work

Since segmentation is typically utilized in a segment-based environment, and most systems available use the frame-based approach, very few segmentation algorithms have

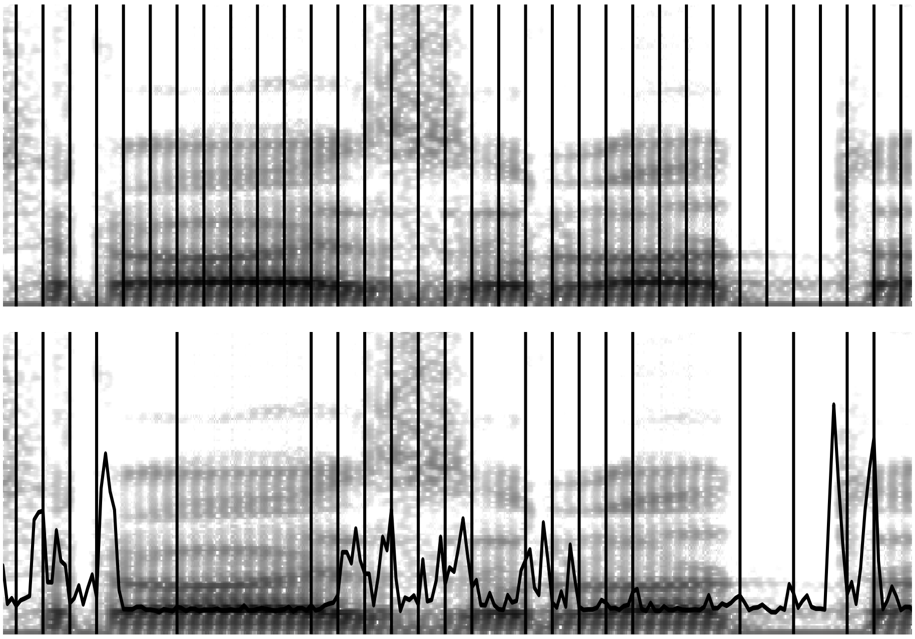

Figure 3.5: Up: the same spectrum as in Figure 3.3 with the output of the basic brute force method. Down: the same spectrum with the $b_i$ values and the output of the Thresholding Algorithm.

been tested to date. The best-known segment-based system, the SUMMIT system of MIT applies the idea of detecting likely bounds between phonemes (these are called *landmarks*), which is done based on the change in the MFCC values. Moreover, each landmark is assigned a probability which is then aggregated to the probability of the hypotheses [31; 32]. It is not clear, however, what the efficiency of this method is, because it cannot be really separated from the search method.

Later further attempts were made to improve this acoustic segmentation in the SUMMIT system. A Viterbi search-based method was proposed, which suggested boundaries based on phoneme boundaries in likely hypotheses, which resulted in a $30\%$ speed-up with the same level of recognition accuracy [70]. Also, segmentation algorithms based on the properties of voiced and unvoiced parts of speech were proposed, with roughly the same performance [87]. The drawbacks of all these methods is that they cannot be strictly separated from the search part, and that they are very close to the signal processing level, which makes their adjustment rather hard if the environment changes (e.g. with noisy speech).

A similar but different concept is the segmentation of speech into words to aid recognition [105].

### Phoneme Boundary Detector Algorithms

In a segment-based speech recognition task it is vital that all actual phoneme boundaries (or at least frames in their immediate neighbourhood) be in the set of possible segmentation bounds $T$. On the other hand we should limit the size of this set, oth-

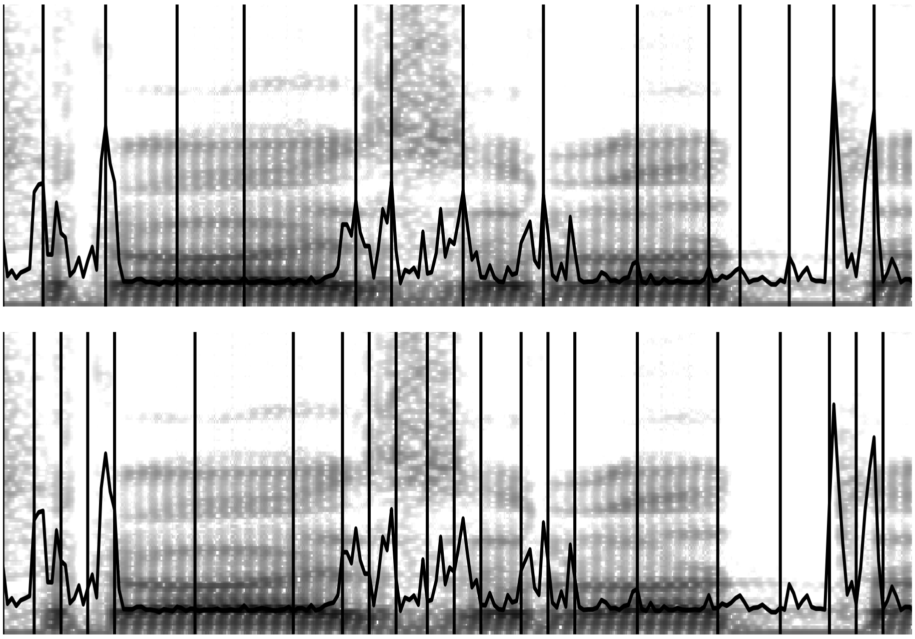

Figure 3.6: Up: the same spectrum as in Figure 3.3 with the $b_i$ values and the output of the Maximum Algorithm. Down: the same spectrum with the $b_i$ values and the output of the LIF Algorithm.

erwise it would lead to an unacceptable loss in speed. This makes the choice of the elements of this $T$ quite important.

A baseline phoneme boundary assignment can be simply to draw one possible phoneme boundary at every $k^{th}$ frame. (See the upper panel of Figure 3.5.) It is a brute force solution, but it leads to quite satisfactory recognition scores. However, using this, during search we have to consider lots of hypotheses, which naturally slows down the search process. On the other hand, this way we cannot miss any real phoneme boundary (in the sense that a possible boundary will be present in its near neighbourhood). But this method is by no means optimal, so next we will describe the algorithms we constructed for phoneme boundary detection.

All our algorithms will rely on a function which estimates for each point the probability of this point being a boundary position. One might think that supplying this value for all frames (all $a_i$ values) solves the entire segmentation problem, but unfortunately this is not the case. While it is indeed a very important factor, in our experiments we found that determining the actual position of the segment bounds is by no means straightforward. So in the following we shall assume that for each frame $a_i$ we have a value $b_i$ which tells us the probability of a phoneme boundary being at that particular place. To get the $b_i$ values we applied ANNs, which will be discussed later in detail.

### Thresholding Algorithm

The first algorithm introduced improves the default brute force segmentation method by applying a threshold. That is, we still draw a possible phoneme boundary at every

$k^{th}$ frame, but only if the corresponding $b_i$ value is greater or equal to some minimum probability $p_{\min}$. The lower panel of Figure 3.5 shows how this method works. One can see that it draws equidistant boundaries everywhere, apart from two intervals where the $b_i$ values are consistently low. Compared to the correct segmentation shown in Figure 3.3, it is clear that this method draws lots of unnecessary boundaries.

### The Maximum Algorithm

Our second algorithm looks for positions where the $b_i$ values take their local maxima over a given interval of neighbouring points. The advantage of this method is that the boundary positions suggested by it in many cases fall pretty close to the ones proposed by humans, as the human experts who manually process such databases also place the boundaries at the points where the spectral change is the largest. By adjusting the size $k$ of the neighbourhood examined by the algorithm we can easily guarantee a minimal distance between the hypothesized boundaries. The algorithm can also be combined with the thresholding method described above. The behaviour of this method is illustrated in the upper panel of Figure 3.6.

### The LIF Algorithm

The problem with both the former algorithms is that we have no direct control over the density of the boundary hypotheses, so in certain cases they may not draw any boundary for a 'long' period of time. A less risky solution would be a modification of the baseline algorithm which puts boundary lines everywhere along the time axis, but dynamically adjusts the density of the markers in such a way that it is proportional to the local probability values $b_i$. A self-evident biologically motivated solution for this is to apply a leaky integrate-and-fire (LIF) neuron model [30]. This construct is the best-known example of the family of spiking neuron models, and its operation is very simple: the neuron integrates its input until the sum reaches a certain threshold. Then the neuron fires (emits a spike), its potential is reset, and the whole process starts again. The model can be refined by making the integration "leaky", in which case the membrane potential decays with a characteristic time constant when no input is present. It may also incorporate an absolute refractory period, above which it is not excitable.

In our segmentation algorithm the $b_i$ values are used as input for the integrate-and-fire neuron. Obviously, where these values are higher, their sum reaches the activation threshold sooner, and the spikes become more dense. With the tuning of the threshold one can control the density of the spikes, while by setting the refractory period a minimum distance between the neighbouring spikes can be easily guaranteed. In our experiments these parameters ($e$ and $k$, respectively) were tuned manually; we tried to set them so that at the most dense areas the density of the markers were similar to those of the baseline algorithm, while at other places the boundary markers were more sparse. The behaviour of this method is illustrated in the lower panel of Figure 3.6.

### Tests

As we mentioned in Chapter 2, we found the best way of measuring the correctness of a possible segmentation (and hence the algorithm which produces it) is to perform word or word sequence recognition tests while applying it. For this we performed a thorough test involving sentence recognition, and examined how the recognition scores varied. For the measurement of the recognition rate, we also used the techniques described there, i.e. accuracy and correctness. The running speed of the recognition process using a segmentation $T$ was expressed as the speed-up achieved via applying it, which is in inverse proportion to the actual running speed compared to the baseline speed.

We used the multi-stack decoding algorithm for searching with a constant big stack size. Although it is true that this parameter also affects both the running speed and the recognition scores, now we wanted to test just our segmentation algorithms. Their performance can be most easily measured by fixing all the other factors, which means the same stack size throughout all the tests; this way the changes both in speed and in the recognition accuracy can be only due to the segmentation method used.

For testing, the Medical Database (see Section 2.5.3) was used, with the task of sentence recognition. We applied a segment-based model using the MFCC-based feature set [45]. To get a (baseline) segment-boundary hypothesis we used the brute force method with $k = 8$, with which we attained, at the word-level, correctness and accuracy scores of $96.42\%$ and $95.34\%$, respectively. These values then served as our baseline. The results of this experiment were published in [44].

### Detecting Phoneme Boundaries

For phoneme boundary detection, first a function which generates the $b_i$ values is required. We trained an ANN to supply these values. As the phoneme boundary positions correlate well with the local spectral changes, we used the first derivates ("$\Delta$ values") of the Mel-Frequency Cepstral Coefficients (MFCC) [50] as features. For each phoneme in the train database we assigned a value of $1$ to the first and last frames, a value of $0$ to the middle frame, and a value between $0$ and $1$ to the rest of the frames depending on how close they were to the sides. Next, the ANN was trained for these frames and target values in regression mode. This way evaluating this ANN on any frame $a_i$ generated the corresponding phoneme bound probability value $b_i$. For an actual example, see Figure 3.4 (both panels).

### Results for the Thresholding Algorithm

Surprisingly, this algorithm did not produce good results. Although with some parameters (like $k = 6$, $p_{\min} = 0.065$) we were able to get a small speed-up, it also caused a decrease in the recognition scores. With some other parameters (like $k = 6$, $p_{\min} = 0.070$) the recognition performance was better than that of the baseline, but this configuration was slower as well. Table 3.7 shows the interesting test results we obtained.

| $k$ | $p_{\min}$ | Correctness | Accuracy | Speed-up rate |
|---|---|---|---|---|
| 6 | 0.065 | 96.06% | 94.94% | 110.83% |
| 6 | 0.070 | 97.49% | 95.70% | 87.70% |
| baseline | | 96.42% | 95.34% | 100.00% |

Table 3.7: Results obtained using the Thresholding Algorithm on the Medical Database (sentence recognition, segment-based context), with the parameters $k$ and $p_{\min}$.

| $k$ | Correctness | Accuracy | Speed-up rate |
|---|---|---|---|
| 3 | 93.90% | 90.68% | 78.29% |
| 4 | 95.34% | 92.83% | 137.02% |
| 5 | 93.19% | 89.96% | 204.79% |
| 6 | 82.80% | 78.49% | 280.11% |
| baseline | 96.42% | 95.34% | 100.00% |

Table 3.8: Results obtained using the Maximum Algorithm on the Medical Database (sentence recognition, segment-based context), with the parameter $k$

Usually the method was just too slow, or when this was not the case, the two recognition percentage scores fell dramatically. These results are probably due to the fact that the phoneme boundary probability estimator ANN was trained to detect the amount of change in the spectrum. Sometimes, however, this change is not abrupt, but occurs over a long time, resulting in just slightly higher $b_i$ values over a longer period. The Thresholding Algorithm was not able to detect these changes, unlike the other two which, in contrast, could take the context of a given frame into account.

### Results for the Maximum Algorithm

The overall results of this algorithm are somewhat mixed. Visually inspecting the possible segmentations it produced this method seemed to be the most promising one, but in practice this was not so. The explanation is that although the method usually inserted very few needless possible bounds, it also skipped some necessary ones, and in practice failing to find phoneme bounds is a very serious mistake. The results can be seen in Table 3.8.

Moreover, the algorithm was tuneable only to a very limited extent: its parameter had to be a small integer, so while with the value of $3$ it was slower than the basic method, even with $5$ it produced a much lower accuracy value. With the only value left in between $(4)$ the accuracy was also somewhat low ($92.83\%$ vs. the baseline $95.34\%$; a $53\%$ increase in the relative error score).

### Results for the LIF Algorithm

This algorithm, unlike the others, achieved a satisfactory improvement in speed with no loss in accuracy (see Figure 3.7). Moreover, it was able to raise the recognition figures

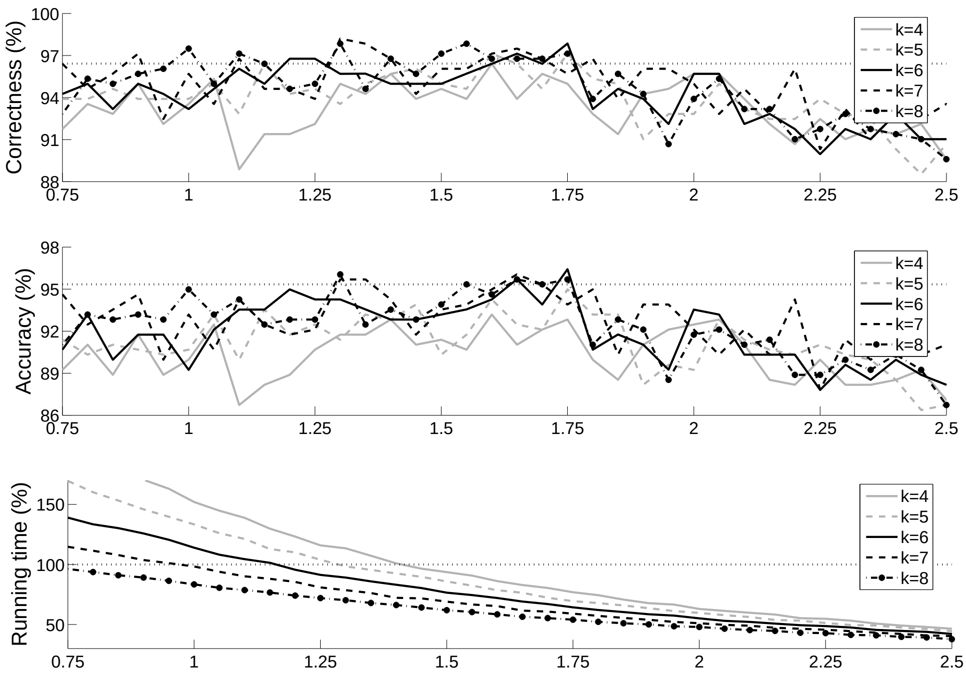

Figure 3.7: The correctness, accuracy and running time results of the LIF algorithm with different $k$ and $e$ values on the Medical Database (segment-based context). The baseline values are represented by a dotted line.

along with a decent gain in speed. This was probably due to the fact that the algorithm was better than the other two in some respects. While the Maximum Algorithm was very good at finding possible phoneme bounds, it could not be tuned. On the other hand, the Thresholding Algorithm could be fine-tuned with its two parameters, but the method itself was too simple to produce really good results. The LIF algorithm, however, seems to be the golden mean between the two: it also has two parameters for tuning, suggests phoneme bounds at much smarter places than the Thresholding Algorithm does, and it takes the context into account.

As for the increase of the recognition scores, it was a surprising result, which was probably due to the combined effect of many factors. Firstly, the phoneme bounds were placed at more accurate positions than they were with the basic method. Secondly, fewer possible phoneme bounds were suggested, thus fewer unusable hypotheses were generated, which did not fill the priority queues that much (as the stack size was the same in each case). And thirdly, due to the fewer possible phoneme bounds being used, it was possible to draw them more closely to each other (i.e. to use a smaller $k$ value) than in the basic method. (The last statement, of course, is true for all three segmentation algorithms.) Some interesting values are shown in Table 3.9, the best values being presented in **bold**.

From the values, the tradeoff between the recognition scores and the running speed can clearly be seen: if we aim for a $100\%$ plus improvement in terms of speed-up, it will eventually lead to a small decrease in both correctness and accuracy. Since accuracy is more important than correctness, we recommend the configuration with the values

| $k$ | $e$ | Correctness | Accuracy | Speed-up rate |
|---|---|---|---|---|
| 5 | 1.75 | 97.13% | 94.98% | 144.15% |
| 6 | 1.75 | 97.85% | **96.42%** | 155.57% |
| 7 | 1.30 | **98.20%** | 95.69% | 127.26% |
| 7 | 2.20 | 96.05% | 94.26% | **215.15%** |
| 8 | 1.30 | 97.85% | 96.05% | 142.39% |
| baseline | | 96.42% | 95.34% | 100.00% |

Table 3.9: Results obtained using the LIF Algorithm on the Medical Database (sentence recognition, segment-based context), with the parameters $k$ and $e$.

$k = 6$ and $e = 1.75$. With these parameters we were able to achieve correctness and accuracy scores of $97.85\%$ and $96.42\%$, respectively, with a speed-up rate of as high as $155.57\%$. These recognition performances mean a $40\%$ and $23.28\%$ improvement in relative error terms, while the speed-up achieved (which results in an actual $35.72\%$ reduction in running times) is also significant.

## 3.4 Summary

In this chapter we focused on the various search issues in the speech recognition problem. First we gave a definition of the search subtask itself, and described the common algorithms used to resolve this problem. Of course these are all well-known issues, which serve only as the basis for the novel ideas introduced by the Author. This chapter covers theses I/1. and I/2., and is covered by publications [39; 40; 44; 63].

Next, a multi-pass search technique was introduced. The idea of a multi-pass method is not uncommon in speech recognition: it means that we perform the search process in several successive steps. In the first one we exclude the highly unlikely hypotheses by some very fast checks; then, for each next step, increasingly precise examinations are made on the increasingly fewer remaining hypotheses. In our actual implementation we restricted the possible set of phonemes by arranging them into several phoneme sets. This grouping was done by applying a standard clustering algorithm; but as this algorithm strongly depends on the distance between the original elements, a distance function had to be introduced. We defined this distance function based on the phoneme classifier (trained and evaluated on the original phoneme set) using its *confusion matrix*. Furthermore, the clustering algorithm required a stopping criterion, which was introduced as well.

After defining the whole clustering process, it results in a possible phoneme grouping. (As in a typical case we have several alternatives for the stopping criterion, there could be several such groupings.) The former search passes were based on them: just these phoneme groups were used instead of the original phonemes, both during classification and in the dictionary. This way the number of words was reduced, phoneme classification could be done more quickly as there were fewer classes, and the search heuristic used (which was the multi-stack decoding one) could be set to a faster configuration (i.e. smaller stack size). After a pass, only the words present at the end remained in the dictionary, and the next pass was performed using just these words. In the final pass the original phoneme grouping was used along with the original phoneme classifier.

This multi-pass search technique was tested on the Children and on the Telephone Databases. On the first one a speed-up ratio of over $330\%$ was attained, meaning a $70\%$ cut in the actual running speed, while on the second one these values were about $240\%$ and about $58\%$, along with the same level of recognition accuracy. As these are good values on two databases of a quite different nature, this method seems to be very promising.

The other speed-up technique introduced in this chapter was that of a segmentation algorithm. This area is rarely investigated in speech recognition, which is probably due to the fact that it is mostly useful in a segment-based context, whereas most systems available rely on the frame-based one. The main idea behind using a segmentation algorithm is that we restrict the positions in the speech signal where there could be a bound between phonemes (i.e. the end of a phoneme and the beginning of the next one). Needless to say, it should be done carefully: carelessly omitting possible bounds like these could lead to a decrease in the recognition accuracy, while suggesting a lot of unnecessary ones dramatically slows down the recognition process.

Apart from the standard, brute-force solution usually applied for this subtask, we introduced and examined three novel methods. All three strongly depend on some function trying to estimate the probability of each individual frame being a possible phoneme boundary, for which purpose a simple solution was proposed. But even using the same function, the three methods introduced performed quite differently: only the last one could produce a significant speed-up and it also improved the recognition scores. This could be due to the fact that it suggested phoneme boundaries in some "smart" positions, which helped the overall recognition process.

Since there is evidently a tradeoff between speed and accuracy, it is hard to characterize the overall performance of this method by a single number; but to illustrate its potential, in a certain configuration it was able to produce a speed-up ratio of over $155\%$, while the accuracy and correctness scores were improved by $23\%$ and $40\%$, respectively when tested on the Medical Database.

Overall, these two new techniques had two common features. First, they sought to speed up the speech recognition process; and second, they did so without affecting the inner workings of the search method applied. Moreover, they both proved to be successful in tests, resulting in a significant speed-up. Their parameter values, by which these results were obtained, were also described, but the fact that the methods could be fine-tuned to suit our needs is more important in general.

# Chapter 4

# Improving the Multi-Stack Decoding Search Algorithm

*"And I've got to get on with my search for the Fountain of Youth.*
*I've devoted my whole life to it, you know," said the old man proudly.*
*Rincewind looked him up and down. "Really?" he said.*
*"Oh, yes. Exclusively. Ever since I was a boy."*
*Terry Pratchett – Eric*

The choice of search method is a very important issue in speech recognition. Perhaps the most widely-used one is the Viterbi beam search algorithm; in Chapter 3 we found, however, that the multi-stack decoding algorithm (or, in other terminology, $n$-best search) has a very similar performance, which is not surprising since the way they work are also quite similar. This was why we chose the latter one as our basic search method throughout our investigations.

This algorithm, however, is very far from being perfect. As it has a constant stack size throughout the whole search process, this value has to be set to the highest value required for the whole speech signal, which is certainly excessive for most cases. (The same is, of course, true for the Viterbi beam search method, with its constant beam width.) This weak point causes the algorithm to waste a lot of time examining and extending unnecessary hypotheses, which heavily affects the overall speed of the speech recognition process. In this chapter we will discuss possible ways of improving the behaviour of these algorithms by studying this issue.

This chapter relies on the contents of Chapter 2 because it contains all the basic terms and concepts needed for speech recognition. There are also references to Chapter 3 where the exact definition of a hypothesis and the detailed search task can be found, which could sometimes prove useful. Moreover, some tests described in this chapter were combined with the multi-pass search method described in Chapter 3.

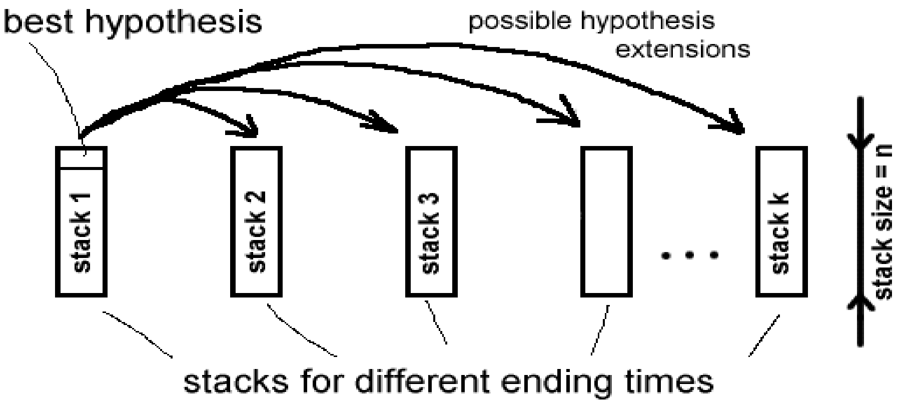
best hypothesis
possible hypothesis extensions
stack 1
stack 2
stack 3
stack k
stack size = n
stacks for different ending times

Figure 4.1: The mechanism of the multi-stack decoding algorithm.

## 4.1 The Properties of the Search Methods Used

As we will focus here on the multi-stack decoding method [3] and we seek to improve its performance, first it is important to describe its behaviour in more detail.

### 4.1.1 The Multi-stack Decoding Algorithm

In the multi-stack decoding algorithm we use the basic data structure of stack (which is not the same as the LIFO- or FIFO-type stack used in other algorithms). A stack is used to store hypotheses (phoneme sequence-segment sequence pairs) ordered by their score, and this score can be either the probability or the cost of the hypothesis. In this algorithm all the stacks are of a limited size; this means that if we try to put more hypotheses into it than is possible, the worst scoring ones (i.e. the ones with the lowest probability or with the highest cost) will be discarded (or *pruned*).

In this algorithm we assign a separate stack for each possible time instance where there could be a boundary between phonemes. Initially all stacks are empty except the first one (belonging to the time instance $t_0 = 1$), which will contain the initial hypothesis $h_0$. Then we examine these stacks in increasing time order: for each step we get all the hypotheses from the current stack, extend them in every possible way (i.e. add a new phoneme and a new segment to its end), and put the generated hypotheses into the stack according to their new ending times. Appendix A.3 gives the pseudocode of the multi-stack decoding algorithm, while Figure 4.1.1 illustrates its basic behaviour.

This algorithm has one parameter, the *stack size*. By altering it, we can control its behaviour: a lower stack size produces a lower accuracy and a quicker run, while a higher value probably leads to a higher accuracy, but the method needs more time to run as more hypotheses will be extended. The weakness of this method is the constant stack size: we use the same value throughout the search process, regardless of the actual stack size needed at the given point of the utterance. If this part of the algorithm could be somewhat improved, its performance should rise as it could find the same ending hypothesis but with a lower running time.

### 4.1.2 Viterbi Beam Search Algorithm

In this chapter we shall also discuss the Viterbi beam search algorithm [69] for two main reasons. First, it can be regarded as a variation of the multi-stack decoding algorithm, so the improvements introduced for it might also prove useful in the Viterbi beam search method. Second, this algorithm is more widely employed in the literature than the multi-stack decoding algorithm; thus if our improvements work well for this method too, they are likely to be more relevant.

In this method the stacks are used in a very similar way to that of the multi-stack decoding algorithm, with only one difference. Instead of assigning the same stack size for each stack, their capacity is controlled differently: each stack contains just the best hypothesis, and some others which are close to it. This "closeness" is defined by their score: a parameter $T$ called the *beam width* is used, and only those hypotheses are kept which are worse than the best one by at most $T$. (For the pseudocode of the algorithm, see Appendix A.4.) The weakness of this method is the same as that of the multi-stack decoding one: throughout the whole evaluation process, the same value of $T$ is used. If a more precise estimate could be made, depending on the context of the actual time instance to which the actual stack is assigned, this algorithm could be made faster while maintaining a high accuracy score.

## 4.2 Techniques for Improving the Multi-stack Decoding Algorithm

When calculating the optimal stack size for the multi-stack decoding algorithm (i.e. the one above for which there is no increase in accuracy), 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 could be significantly improved. (Of course, doing it carelessly could lead to a loss in accuracy, which should be avoided.) We will suggest a number of ideas and then test their effectiveness.

$i)$ 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 in the stack, 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 idea is based on the observation that at the beginning of the utterance we have very little information to decide which hypothesis should be kept, thus we need bigger stacks than we will need later, which can be utilized for a slight speed-up. A simple solution for this was implemented: 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. Needless to say, $m$ should be close to 1, otherwise the stacks would soon be far too small.

$iii)$ Another technique is a well-known modification of stacks. It can easily happen that there are two or more hypotheses which have the same end times and the same phoneme sequence, because some earlier phoneme bound is at a different time instance. In this case it is sufficient to keep only the most probable one as the further extensions rely only on the ending time and the phoneme sequence of a hypothesis, which are the same in this case. The stacks can be easily modified to work this way, but it leads to a time-consuming examination of the content of the stack every time a new hypothesis is inserted. Due to this, we implemented a simpler, approximate solution for this idea: we allow the stacks to store more of these "same" hypotheses, but when extending a hypothesis from the stack, we first pop it and its phoneme sequence is compared to the previously examined one (when there was one, that is). The current hypothesis is extended only if the phoneme sequences differ, otherwise it is simply thrown away. As our preliminary tests suggested that this approach is in general more efficient, we applied this one in the experiments section presented later.

*iv)* Yet another approach for improving the method comes from the observation that we need big stacks only at those segment bounds where they more probably 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 in regression mode for this task on the 13 MFCC $\Delta$ features with the actual segment bounds having a value of 1 and the middle of each phoneme having a value of 0. The output of the neural net was then 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 finishing hypothesis which was chosen, as (due to the large stack size) it is probably the one with the lowest cost. Next we examined the path which led to the winning hypothesis (all its earlier phoneme bounds), and noted the required stack size (the actual position in its stack) and the segment bound probability $p$. The result, represented as a stacksize–probability diagram, was used to get a proper fitting curve for estimating the required stack size [42]. It was shown that most of the higher stack sizes are associated with a high value of $p$, so the stack size can indeed be estimated using this probability value.

In some experiments, 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. Further, some publications contained large-scale experiments which involved not only the application of these improvements, but, at the same time, the multi-pass search detailed in Section 3.3.1 [40; 63]. In these cases it was rather hard to combine 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, a greatly simplified form was applied: if the probability of the given time instance being a bound is lower 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 in most cases it is indeed effective (see the results sections).

### 4.2.1 Testing

We now turn to the testing of these novel speed-up improvements. However, their application is not trivial if we apply several at a time: we could use them in various combinations and some of them have parameters which also affect the accuracy and speed of the search process. For this reason in these cases we used an iterative technique called *sequential forward selection* [19; 89]. First all the improvements are tested separately, which includes the determination of their optimal parameters. In the following we will keep only the one which produced the biggest speed-up along with the given parameter value. In the next step we test the remaining improvements and set their parameters, until we have gone through all the possible combinations [38]. In the tables we always separate the steps of this iteration by a horizontal line. The fastest configuration of an iteration (which, naturally, still produces the required minimal recognition accuracy) is highlighted 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.

### 4.2.2 Testing for the Numbers Database

We performed tests for these improvements in several environments. The first test was made on the Numbers Database (see Section 2.5.3), on both Test Set I and Test Set II, following the segment-based approach with the feature set introduced by Tóth [64]. These were word-level experiments on a rather small database consisting of utterances of Hungarian numbers. In these series of experiments we wanted to carry out a large-scale scan of the standard search algorithms, thus we checked the stack-decoding and A* algorithms too; but not all the improvements were employed on the multi-stack decoding method. Each method was expected to attain the minimum number of hits of 304 of 312 on Test Set I, and 132 of 169 on Test Set II, meaning $97.44\%$ and $78.10\%$ in word hit percentage scores. (These are the optimal values our speech recognition system was able to reach at that time, so we required each search method to hit all the correct words that were possible.) Then the fastest parameter setting which could achieve these percentages was chosen. Although these scores may seem low from the current point of view, they were quite realistic for the Hungarian language at the time of publication (2002). The results of this experiment were published in [42].

Improvement $iii$ was used in every test (even those of the baseline). For the remaining ones, only $i$ and $iv$ were tested, but for the latter one we checked both variations. The simpler one (variation $a$) was the only two-step stack size determination. Variation $b$, however, was a more complex process. To find a function that approximates the necessary stack size based on the output $p$ of the estimating ANN, we carried out the large-scale experiment defined during the description of improvement $iv$. The points plotted in Figure 4.2 show the necessary stack sizes as a function of $p$.

Now, for a phoneme-bound probability $p$ (supplied by the ANN), we fitted a min($c_0+e^{c_1 \cdot p+c_2}$, $c_3$) curve as a stack size. Needless to say, the value for $c_3$ comes from the test of the multi-stack decoding algorithm, and the value for $c_0$ from an examination

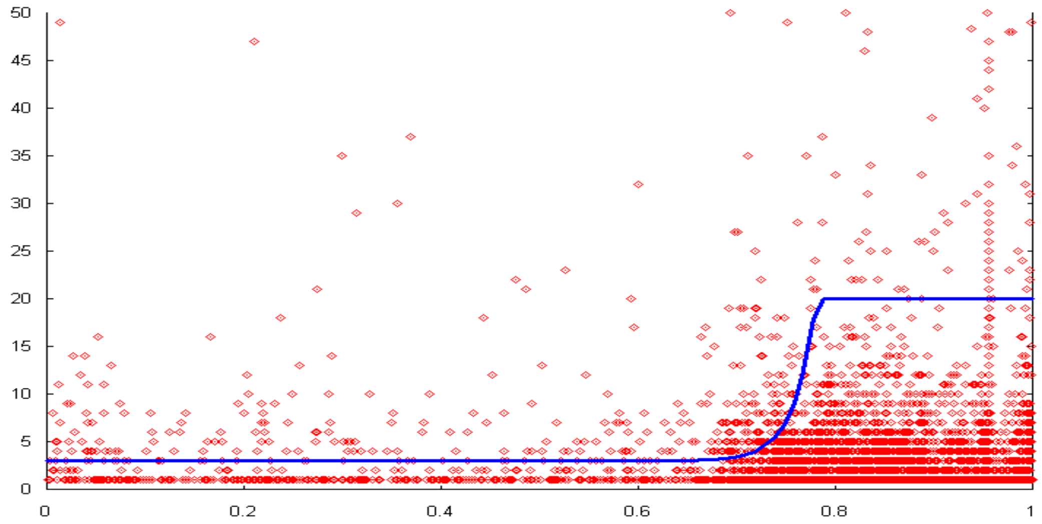

Figure 4.2: Bound probability – necessary stack size diagram with the best fitting curve for the Numbers Database, Test Set I

of variation $a$) (as the lower phoneme bound). For a given $c_1$, $c_2$ can be determined by trial and error. The best fitting curve was plotted in Figure 4.2. For variation $a$), the limit for $p$ where we changed the stack size was finally set to $0.7$.

We did not use the sequential forward selection for these two improvements because the number of possible combinations was so small. Instead, we first used improvement $i$, then we applied variations $a$ and $b$ of improvement $ii$. The speed was measured, as usual, based on the number of phoneme classification ANN calls, as it is linearly proportional to the actual running time (only much easier to measure). Then, as our basic search method was the multi-stack decoding algorithm, the running times were converted into percentage values relative to the multi-stack decoding method. The speed and speed-up values were obtained by taking the reciprocal of these values.

### Results

The results for the methods can be seen in Table 4.1. For the standard methods, the stack decoding algorithm performed surprisingly well on the first test set. Extending the best-scoring of all hypotheses can be regarded as a heuristic, which actually performs quite well with short utterances. Unfortunately, on longer words it proved unsatisfactory: on the second set (whose elements were much closer to real-life examples) it yielded the worst results of all the methods tested here. As for its extension with the above-mentioned $A*$ heuristic, it proved to be unsuccessful for both test sets.

The multi-stack decoding method seems the most promising. Even though it did not perform outstandingly well, it produced fair results and, unlike the other methods mentioned here, there is significant possibility for improvement. However, of all the standard algorithms the Viterbi beam search 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 highest speeds of the

| | Relative speed | |
|---|---|---|
| Method combinations | Test Set I | Test Set II |
| stack decoding | 230.26% | 34.71% |
| A* heuristic | 76.09% | 7.54% |
| multi-stack decoding | 100.00% | 100.00% |
| Viterbi beam search | 157.78% | 102.24% |
| multi-stack + $i$ | 187.83% | 149.26% |
| multi-stack + $i$ + $iv$ variation $a$ | 206.31% | 153.08% |
| multi-stack + $i$ + $iv$ variation $b$ | 231.27% | 165.67% |

Table 4.1: The test results of the improvements applied for both sets of the Numbers Database (isolated word recognition, segment-based context). The running times of each method combination was measured via the number of phoneme classifications; the relative speed ratios achieved are inversely proportional to these values.

four standard methods.

Among the former algorithms only the Viterbi beam search and the multi-stack decoding methods could be combined (the stack decoding and multi-stack decoding methods are fundamentally different, and the A* algorithm is already an improved version of the stack decoding method). Combining the first two methods (i.e. improvement $i$) led to a much more efficient algorithm, especially for Set II.

In order to evaluate the probability of a bound we used an ANN, which classified a bound to 80% accuracy using derivative-like features. For variation $a$, we can say that we were able to achieve a slight speed-up over the improvement $i$ scores, especially on Test Set I. Variation $b$, however, proved to be much better, significantly reducing the running times on both test sets. We can thus say that this novel method is definitely better than the standard algorithms tested here.

### 4.2.3 Testing All Improvements for the Numbers Database

Next we tested all four improvements more systematically, though still on the Numbers Database and still following the segment-based approach. We applied the sequential forward selection algorithm, and employed no implicit improvements as we did with improvement $iii$ in the previous test. As for improvement $iv$, however, we used just the simpler variation (i.e. variation $a$) because the parameters had to be set several times, and for this variation it is much more straightforward. The results of this experiment were published in [37].

The database was equivalent to Test Set I of the previous experiments, i.e. the isolated word recognition of 312 numbers, from which we expected the methods to hit at least 304 (the maximum number of correct hits for our speech recognition system on this database at that time). They all were able to do this, although with different parameter values even for each improvement combination. Running times, as usual, were measured using the number of phoneme-identifications; then the amount of speed-up was calculated, which is proportional to the reciprocal of the running time. For a

| Method combinations | Speed-up ratio |
|---|---|
| Viterbi beam search | *68.65%* |
| multi-stack decoding | *100.00%* |
| multi-stack + $i$ | 165.85% |
| multi-stack + $ii$ | 101.44% |
| multi-stack + $iii$ | **241.33%** |
| multi-stack + $iv$ | 114.36% |
| multi-stack + $iii$ + $i$ | **422.59%** |
| multi-stack + $iii$ + $ii$ | 266.38% |
| multi-stack + $iii$ + $iv$ | 293.14% |
| multi-stack + $iii$ + $i$ + $ii$ | 435.93% |
| multi-stack + $iii$ + $i$ + $iv$ | **515.23%** |
| multi-stack + $iii$ + $i$ + $iv$ + $ii$ | 519.36% |

Table 4.2: The test results of the improvements applied on Test Set I of the Numbers Database (isolated word recognition, segment-based context), with the sequential forward selection technique. The running times of each method combination was measured via the number of phoneme classifications; the speed-up ratios achieved are inversely proportional to these values.

comparison we also performed a standard Viterbi beam search test. The feature set used was the one developed by Tóth [64]. The results can be seen in Table 4.2. It is quite surprising that improvement $iii$ produced the best results. After that, similar to the previous test, improvement $i$ (the Viterbi beam search) reduced the search space the most, followed by improvement $iv$. It can be seen that all the improvements enhanced the recognition performance in every case; the best result was obtained by using all of them together. But we also suggest partial combinations such as combination $iii$ + $i$ + $iv$, having found that a further extension did not really enhance the performance but introducing another parameter to set.

Overall we conclude that, on examining the test results, it is clear that the multi-stack decoding and Viterbi beam search methods can indeed be combined, without any loss of accuracy (since no adjustment of their optimal parameters was needed), and with a marked improvement in performance. Further significant search space reductions were also possible with the other proposed improvements. In the end our new combination ran about 5 times faster than the multi-stack decoding method, and about 8 times faster than the Viterbi beam search method.

### 4.2.4 Testing All Improvements for the Children Database

This test was similar to the previous one, but it was carried out on a database of a more significant size: on the Children Database (see Section 2.5.3). The task was still word recognition on a segment-based basis, but now using the feature set based on the MFCC values [45]. In our tests we expected an $80\%$ word accuracy at least. The results of this experiment were published in [38].

| | | Viterbi | multi-stack | $iii$ | $iii + i$ | $iii + i +$ | $iii + i +$ |
|---|---|---|---|---|---|---|---|
| $\lambda$ | | beam search | decoding | | | $iv$ | $iv + ii$ |
| | **A** | $98.61\%$ | $172.99\%$ | $281.28\%$ | $301.96\%$ | $343.55\%$ | $344.20\%$ |
| 1.0 | **B** | $108.56\%$ | $319.23\%$ | $388.84\%$ | $507.30\%$ | $573.12\%$ | $580.47\%$ |
| | **D** | $60.99\%$ | $160.33\%$ | $274.20\%$ | $274.68\%$ | $342.44\%$ | $366.74\%$ |
| | **A** | $59.60\%$ | $182.65\%$ | $281.74\%$ | $283.46\%$ | $346.27\%$ | $347.14\%$ |
| 0.9 | **B** | $98.54\%$ | $319.30\%$ | $389.27\%$ | $503.77\%$ | $550.17\%$ | $553.16\%$ |
| | **D** | $66.04\%$ | $213.00\%$ | $274.02\%$ | $305.06\%$ | $359.87\%$ | $379.23\%$ |
| | **A** | $48.69\%$ | $182.65\%$ | $234.90\%$ | $238.76\%$ | $256.69\%$ | $262.39\%$ |
| 0.8 | **B** | $84.00\%$ | $255.55\%$ | $350.53\%$ | $429.07\%$ | $458.43\%$ | $461.11\%$ |
| | **D** | $65.40\%$ | $182.63\%$ | $228.43\%$ | $265.19\%$ | $286.77\%$ | $291.33\%$ |
| | **A** | $29.04\%$ | $107.72\%$ | $130.06\%$ | $130.49\%$ | $133.62\%$ | $135.85\%$ |
| 0.7 | **B** | $73.17\%$ | $255.57\%$ | $351.09\%$ | $426.51\%$ | $448.44\%$ | $450.63\%$ |
| | **D** | $65.93\%$ | $182.65\%$ | $228.49\%$ | $245.90\%$ | $270.10\%$ | $272.97\%$ |

Table 4.3: The test results for the improvements applied after using the various aggregation operators on the Children Database (isolated word recognition, segment-based context). The speed of each method combination was measured via the number of phoneme classifications; the speed-up ratios achieved are inversely proportional to these values.

In this test, however, we did not just seek to test the speed-up improvements. It was an investigation which also involved the use of aggregation operators, which were used to improve the recognition scores. We will elaborate on them in Chapter 5, but now we will treat their application as a black box. What is important for us is that there were four variations (**A**, **B**, **C** and **D**), and their behaviour was governed by a $\lambda$ parameter. Variation **C** could not attain the $80\%$ minimum, so we tested the remaining three with $\lambda$ values of $0.7$, $0.8$, $0.9$ and $1.0$, resulting in a much larger number of tests. First one of these operator variations were used, then the right parameter values of the search methods were determined to achieve the minimal accuracy. So, from the viewpoint of our speed-up improvements, this stage played the role of baseline; the only inconvenience the aggregation operators caused was that there were $12$ different cases.

For the sake of simplicity, we used the same order of improvement-application for all twelve cases, which was arranged in the same way as the test results for the second test for the Numbers Database: first the "same" hypotheses were discarded ($iii$); then the two basic, stack-based search techniques were combined ($i$); next the stack size was reduced for stacks at unpromising phoneme boundaries ($iv$); finally, advancing in time, we reduced the stack sizes ($ii$). In this test series, owing to the large numbers of experiments, we still applied the simpler variation of improvement $iv$ (variation $a$).

The speed of a speech recognition configuration was still measured based on the number of phoneme identifications (i.e. ANN calls), then it was converted to a speed-up rate which is inversely proportional to the running times (see Table 4.3). The reference value (i.e. $100\%$) was the speed of the multi-stack decoding algorithm without using any kind of operator, but perhaps the relationship between the speed of a multi-stack decoding search using an aggregation operator and the speed of the corresponding

| Method combinations | Speed-up ratio |
|---|---|
| Viterbi beam search + $iii$ | *114.97%* |
| multi-stack decoding + $iii$ | *100.00%* |
| multi-stack + $iii$ + $i$ | **554.80%** |
| multi-stack + $iii$ + $ii$ | 412.20% |
| multi-stack + $iii$ + $iv$ | 436.30% |
| multi-stack + $iii$ + $i$ + $ii$ | 687.28% |
| multi-stack + $iii$ + $i$ + $iv$ | **702.24%** |
| multi-stack + $iii$ + $i$ + $iv$ + $ii$ | 809.60% |

Table 4.4: Performance of the basic search methods (the Viterbi beam search and the multi-stack decoding algorithm), and the applied improvements and improvement combinations, with the sequential forward selection technique, on the Telephone Database (isolated word recognition, frame-based context). The speed of each method combination was measured via the number of phoneme classifications; the speed-up ratios achieved are inversely proportional to these values.

speed-up configurations (i.e. values within the same line) could be of more interest. A significant improvement could be attained compared to these baseline values (4-6 times faster than the Viterbi beam search, 1.5-2 times faster than the multi-stack decoding algorithm), but it strongly depends on the environment: the aggregation operator used.

### 4.2.5 Testing All Improvements for the Telephone Database

The last test we performed was made on the Telephone Database, which is also of significant size. We again performed a word recognition task. This experiment was also quite a complex one: first, similar to the test on the Children Database, aggregation operators were used (see Section 5.1); then multi-pass tests were carried out (see Section 3.3.1); and finally we carried out the speed-up improvements described above. The aim of this testing procedure was to demonstrate that all these techniques could be applied altogether, which was indeed proven. From our current point of view, however, the only interesting thing is that, similar to the test on the Children Database, the speed-up improvements were utilized in several different environments. The results of this experiment were published in [63].

The *HTK* system [111] was used as a reference and produced a score of $92.11\%$. Our testing was done with a frame-based model, with the features of the standard 39 MFCC + $\Delta$ + $\Delta\Delta$ coefficients and using Artificial Neural Networks. First the aggregation operators were employed, then we performed the multi-pass tests. At the end of each multi-pass step we had a configuration of stack sizes and passes which attained a word accuracy of at least $94\%$.

In the multi-pass tests we performed a search for the hypothesis with the lowest cost in several steps, each one examining the well-performing hypotheses of the previous pass only, but doing it more precisely. We did it by restricting the phoneme set, which could be done in 4 possible ways (either by using the distance function $d^1$ or $d^2$, and

| Phoneme group | | Passes | | | Applied Improvements | | | |
|---|---|---|---|---|---|---|---|---|
| | | $\mathcal{P}_0$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $iii$ | $iii + i$ | $iii + i + iv$ | $iii + i + iv + ii$ |
| standard | | ● | ○ | ○ | 100.00% | 554.93% | 702.24% | 809.06% |
| $d^1$ | $\mathcal{D}_{\min}$ | ● | ● | ○ | 174.82% | 824.40% | 1017.29% | 1037.34% |
| | | ● | ○ | ● | 160.97% | 251.00% | 303.03% | 315.55% |
| | | ● | ● | ● | 113.30% | 214.50% | 267.66% | 282.96% |
| | $\mathcal{D}_{\max}$ | ● | ● | ○ | 241.25% | 945.17% | 1193.31% | **1253.13%** |
| | | ● | ○ | ● | 216.21% | 442.08% | 512.55% | 548.54% |
| | | ● | ● | ● | 184.70% | 374.39% | 481.23% | 501.25% |
| $d^2$ | $\mathcal{D}_{\min}$ | ● | ● | ○ | 204.62% | 874.89% | 1095.29% | 1154.73% |
| | | ● | ○ | ● | 151.92% | 346.62% | 477.09% | 500.25% |
| | | ● | ● | ● | 126.18% | 287.60% | 359.97% | 423.72% |
| | $\mathcal{D}_{\max}$ | ● | ● | ○ | 203.12% | 777.60% | 872.60% | 927.64% |
| | | ● | ○ | ● | 239.63% | 566.89% | 679.34% | 703.23% |
| | | ● | ● | ● | 168.32% | 545.55% | 693.00% | 732.60% |

Table 4.5: Performance of the multi-pass search configurations, combined with the improvements, on the Telephone Database (isolated word recognition, frame-based context). The ● symbol means we applied the given pass in the given configuration, while the ○ symbol means that we omitted it. The percentage values represent the speed-up ratio achieved, which is inversely proportional to the actual running times.

either by using the other distance function $\mathcal{D}_{\min}$ or $\mathcal{D}_{\max}$). We constructed two earlier passes for each of the four possible ways (denoted by $\mathcal{P}_1$ and $\mathcal{P}_2$, the original pass being $\mathcal{P}_0$), among which any could be used, or both of them. These 3 variations led to 12 possibilities overall.

In order to find a good-performing order of the speed-up improvements, we had to turn to the sequential forward selection technique again. We applied it with a simple multi-stack decoding search on the original phoneme set and the resulting ordering was applied to each multi-pass configuration. Note that in this test we did not test improvement $iii$, but we applied it to each test case instead (including the baseline values and all the multi-pass steps). Also, similar to the test for the Children Database, we applied the simpler variation of improvement $iv$ (variation $a$).

The results for this sequential forward selection test can be seen in Table 4.4, where the iterations are separated by a horizontal line. The fastest configuration obtained for an iteration was highlighted in **bold**, while the order of the improvements represents the order of fine-tuning the parameters. Thus the tested configurations in an iteration differ only in the last noted improvement. The speed-up rate was calculated as before, so it is inversely proportional to the actual running times.

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

If we examine the results in Table 4.5, it can be seen that with these modifications the search process could indeed be speeded up significantly. But it is also apparent 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: the two-pass search method 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.

## 4.3 Summary

In this chapter, similar to the previous one, we focused on the search subtask of the speech recognition process. This time we concentrated on the multi-stack decoding algorithm (also known as $n$-best search in the literature), which was the basic search technique used throughout our investigations. This chapter covers Thesis I/3, and is covered by publications [37; 38; 40; 42; 63].

This search method is quite effective even in its basic form; however, it has a handicap. It assigns a data structure called a stack to every possible location where there could be a bound between phonemes, but these stacks have the same capacity regardless of their environment, which could be the cause of a significant speed loss. Thinking that using constant-sized stacks is the main weak point of this algorithm, our efforts in this chapter concentrated on suggesting smarter ways of estimating the sufficient size for each stack individually. For it we introduced four novel approaches, which, of course, can be combined: then each stack will have the lowest capacity suggested by these improvements. Some improvements introduced have parameters which have to be set, which is not really feasible if more improvements are simultaneously applied. To resolve this issue we applied the standard technique called sequential forward selection: first all improvements were tested separately, then the best-performing one was chosen and was applied throughout the remaining tests with its determined parameter value. Next, the remaining improvements were tested again to determine their now-optimal parameters, and so on.

Our improvements were tested on the Numbers Database (twice, on two different test sets), on the Children Database and on the Telephone Database; as their aim was to get a speed-up, we expected the recognition accuracy to remain at the same level as it was before. Under these circumstances we were able to attain a speed-up ratio of about $300 - 500\%$ on the Numbers Database, about $340 - 580\%$ on the Children Database, and above $800\%$ on the Telephone database (meaning $33 - 80\%$, $71 - 83\%$ and $86\%$ reductions in the actual running times for these databases, respectively). Our tests showed that all the improvements are worthwile (though not to the same degree), and that they could indeed be combined, even with techniques described in Chapter 3.

# Chapter 5

# Using Aggregation Operators to Calculate Hypothesis Probabilities

*"Well, all right, last desperate million-to-one chances always work, right,*
*no problem, but... well, it's pretty wossname, specific. I mean, isn't it?"*
*"You tell me," said Nobby.*
*"What if it's just a thousand-to-one chance?" said Colon agonisedly.*
*"What?"*
*"Anyone ever heard of a thousand-to-one shot coming up?"*
*Carrot looked up. "Don't be daft, Sergeant," he said. "No-one ever saw a thousand-to-one*
*chance come up. The odds against it are -" his lips moved - "millions to one."*
*Terry Pratchett – Guards! Guards!*

The Reader should recall that in the subtask of hypothesis probability calculation there are two distinct aggregation levels (see Section 2.3.1 in Chapter 2). First the probability of a given phoneme is estimated on the given segment based on local information sources, whose value is supplied by some $g_1$ operator. Then these phoneme-level values are aggregated to a hypothesis-level score with the application of a $g_2$ operator. This $g_1$ does not always involve an actual operator; for example in a typical segment-based context it is just the length-normalized output of the phoneme classifier (treated as a probability value). In a standard frame-based environment, however, it does, as it is the product of the frame-level phoneme classification scores. As for $g_2$, it is practically always an operator as it is basically the product of the values supplied by $g_1$.

(Note that in this brief summary, for the sake of simplicity, we intentionally omitted some details like the transition probabilities; but we think that these minor points do not really affect our general reasoning.)

An operator used in a certain context is always just a model of the underlying events as we cannot perfectly simulate the actual inner workings of the problem being modelled. For example, the product of the frame-level probability likelihoods in $g_1$ corresponds to assuming a high degree of independence between the consecutive frames in the speech signal, while in the case of $g_2$ we assume that the consecutive phonemes are independent. Though this assumption of independence leads to a convenient mathematical formulation, and it also behaves quite well in practice, this assumption is, of course, false owing to the continuous motion of the vocal chords, the tongue, the mouth and so on. [112]. Thus there is no good reason to stick to multiplication on either of these levels except for its performance; but if we could find other operators with the same level (or even higher) performance, their application could be just as legitimate, as it would just mean that we simply use other assumptions. In this chapter we will focus on this issue.

Naturally it does not mean that *any* operator could (or should) be used. There are indeed some criteria which are straightforward to set. Perhaps the most trivial one is that it should be an operator, which means that it must accept any number of arguments. The other requirement might be that of easy use: the integration of the actual operator into either of these levels should be done fairly easily by requiring as few modifications on either side as possible. And the third requirement expected is tunability: the behaviour of the operator applied should be easily adjustable to make it fit a given problem, which can be done most easily by using operators with one or more parameters.

In this chapter we shall be primarily interested in improving the seemingly arbitrary step of replacing the simple products of speech-phoneme probability likelihood values by some other operators at both levels. First we shall experiment with self-consistent mean operators taken from the field of fuzzy logics; we will also introduce a variant called the decaying root-power mean operator. Then we will turn to the application of triangular norms, which are the conjunctive operators of fuzzy logic. The most plausible choice is to apply some well-known t-norms in this problem; then, driven by the hope of finding a better-performing operator, we will examine a more general triangular norm and test the Generalized Dombi Operator family. (Of course all of these are operators taken from the literature.) Finally we will look for a much more general representation of triangular norms by modelling procedure their generator function. For this reason the logarithmic generator function is introduced, and a method is presented to finely apply this modeling to the actual task. Note that this method is of a general character, so there is no reason to limit its application to just that of speech recognition.

The structure of this chapter is somewhat mixed. All the steps mentioned above have been placed into a separate section, but as all these steps require some prior knowledge, we also have to describe them. Thus each section begins with a description of some well-known concepts found in the literature. Then in the rest of the sections we deal with the ideas or novel methods we introduced, or the solutions we came up with for the problems we faced like the application of some norms in some context. At the end of each section we describe the tests done, present and analyze the test results.

## Related Work

There are many areas in the literature where fuzzy operators are utilized in the speech recognition problem. A large amount of effort has been made in applying fuzzy techniques in phoneme recognition, including using fuzzy neural networks [57], fuzzy clustering [103] and fuzzy rules [9]. Other areas include the fuzzy Expectation Maximization algorithm to set the state transition values of a HMM [1; 102] and "fuzzy match" methods for detecting out-of-vocabulary (OOV) words [91]. The application of fuzzy operators for cost/probability value aggregation, however, can be considered novel.

The reason for it is probably that even this approach of estimating the probability score on this two-level decomposition (with the use of $g_1$ and $g_2$) is a rather new idea, being published only recently [99]; also, it is not trivial, especially from a HMM viewpoint (which is the most commonly used approach in speech recognition). Furthermore, little attention is paid to the inner mechanisms of a HMM. Thus, although quite a lot of research has been done in applying fuzzy techniques in the speech recognition problem, they practically neglect the cost/probability aggregation subtask.

## 5.1 Using Mean Aggregation Operators

In our first experiments we looked for an operator with a wide tunability between entirely different behaviours; in the end we turned to the set of mean aggregation operators. By the term "operator" we will follow the fuzzy terminology: we mean something which differs from a function only in one property: it takes any number of arguments. As the mean aggregation operators have this property, and there are some which can be easily tuned between wide bounds, we opted for this set.

### 5.1.1 Mean Aggregation Operators

The term *mean aggregation operators* is well known in fuzzy literature [60]. We will use the definitions of [24], but to use them in our actual problem, we have to make some modifications to them. During the speech recognition process very small probability values appear, which, due to computer arithmetic, tend to become real 0 values, making the whole speech recognition process impossible. To overcome this problem, instead of a probability $p$ usually the cost value $c = -\log p$ is used; these values are then added up (instead of multiplying the original probabilities), and finally the lowest one is sought. The operators used should fit into this scheme, which means that they (in their basic form, or a modified version) should work with cost values both as their input and output. The mean aggregation operators, however, work only with values in the $[0, 1]$ interval in their basic form, so we have to extend the terms found in [24] to meet this requirement and make them able to handle the cost values as well.

**Definition 5.1** *A mapping $G : [0, \infty)^j \rightarrow [0, \infty)$ is called a mean aggregation operator if it satisfies the following conditions:*

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

Next we seek to define an operator class which takes any number of arguments; for it, first we need the concept of a *bag* (or *multiset* in other terminology). 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,

$$\mathcal{B}^{[0,\infty)} = \bigcup_{j \geq 1} [0, \infty)^j. \tag{5.1}$$

**Definition 5.2** *A mapping $G : \mathcal{B}^{[0,\infty)} \longrightarrow [0, \infty)$ is a self-consistent mean operator if $G$ satisfies the following conditions:*

1. $G(x) = x$ (naturalness)
2. $G$ is indifferent to the order of the arguments (commutativity)
3. for bags with the same dimension condition, *M2* applies (monotonicity)
4. $G(x_1, \ldots, x_j, e) = G(x_1, \ldots, x_j)$ if $e = G(x_1, \ldots, x_j)$ (self-identity)

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

$$G_\alpha(x_1, \ldots, x_j) = \left( \frac{x_1^\alpha + \ldots + x_j^\alpha}{j} \right)^{\frac{1}{\alpha}}, \quad \alpha \in \mathbb{R}. \tag{5.2}$$

in our task. It is well-known [16; 47], that when $\alpha \rightarrow -\infty$, $G_\alpha \rightarrow \min(x_1, \ldots, x_j)$; $G_{-1}$ equals the harmonic mean; when $\alpha \rightarrow 0$, $G_\alpha$ tends to the geometrical mean; $G_1$ equals the arithmetical mean; and when $\alpha \rightarrow \infty$, $G_\alpha \rightarrow \max(x_1, \ldots, x_j)$. By varying the $\alpha$ parameter we can make a continuous transition from the minimum operator to the maximum operator, which means that it is quite flexible, a property which might come in handy in any application. Next we define a variant of the root-power mean operator, the *decaying root-power mean operator* as

$$G_{\alpha,\lambda}(x_1, \ldots, x_j) = \left( \frac{\lambda^{j-1} x_1^\alpha + \lambda^{j-2} x_2^\alpha + \ldots + \lambda x_{j-1}^\alpha + x_j^\alpha}{j} \right)^{\frac{1}{\alpha}}, \tag{5.3}$$

where $\alpha \in \mathbb{R}$ 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 cost of the $i$th phoneme on the $i$th segment, while $\lambda^{j-i}$ is a weighting factor for $x_i$; this way, advancing in time, the cost of earlier phonemes will become less and less dominant in the aggregation form. We expected this modification to perform well due to the typical search methods used in speech recognition, namely the multi-stack decoding algorithm and the Viterbi beam search. These methods advance in time and keep only the best hypotheses, i.e. the ones with the lowest cost. These hypotheses were already among the best ones previously, only without their last phoneme; thus the last phoneme may require special attention, whereas the others should receive smaller weights. Note that this variant itself is not a self-consistent mean operator.

### 5.1.2 Tests

Testing an operator with just one parameter is by no means a complicated task. First we run preliminary tests: with trial parameter values we determine the region where the parameter makes the operator work well in the desired task. Then, given the upper and lower bounds of this region, we run several tests automatically with a step size small enough to explore this area. Finally we can examine the results of these tests, and note the best accuracy values along with the parameter values which produced these. Since the original root-power mean operator has just one parameter ($\alpha$), we could easily test it this way. The decaying root-power mean operator, however, has another parameter: $\lambda \in [0,1]$. To overcome this problem we performed our tests with only some selected $\lambda$ values; then, of course, the remaining $\alpha$ parameter could be tested as described above.

Applying these kinds of operators in speech recognition, though, may require some small modifications. The basic operator we would like to change is addition, but, unfortunately, the root-power mean operator does not include this operation: only the arithmetical mean is covered. But it seems logical that the new operator applied should work in a similar way to the one we replaced it with. To overcome this contradiction, we have two plausible possibilities:

$$G^1_\alpha(x_1,\ldots,x_j) = j \cdot G_\alpha(x_1,\ldots,x_j) \tag{5.4}$$

and

$$G^2_\alpha(x_1,\ldots,x_j) = (x_1^\alpha + \ldots + x_j^\alpha)^{\frac{1}{\alpha}}. \tag{5.5}$$

Thus, in the first case the result of the operator applied is simply multiplied by the number of arguments, while in the second one the inner workings of the operator is modified by omitting the division by $j$. We can do this because we no longer require that our operators work in the $[0,1]$ interval.

#### Testing in a Segment-based Case

In a segment-based environment $g_1$ is typically a length-normalized output of some statistical machine-learning algorithm, so there is no way we could use aggregation operators at this level. But if we apply such an operator at a higher level (i.e. as $g_2$), it may happen that it works better if we omit this length-normalization. There is no way to know which is best other than by testing, so we will test both variations.

On the other hand, for $g_2$, the operator can easily be applied since at this level the default operation is the sum of the costs appearing as $g_1$. It is not certain, however, whether the application of the original root-power mean operator would produce better results, or the use of $G^1_\alpha$ (or perhaps $G^2_\alpha$) would. This leads to four possible test types (where **D** with the $\alpha = 1$ value means conventional addition):

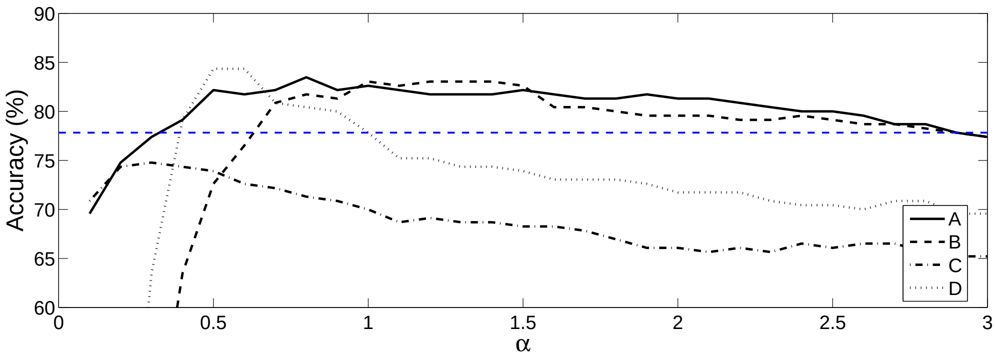

Figure 5.1: Recognition accuracy for the Children Database (isolated word recognition, segment-based context) with aggregation operator variations **A**-**D** ($\alpha \in [0.1, 3]$, $\lambda = 1.0$) as $g_2$. The baseline value is represented by the horizontal dashed line.

| **Variation** | $g_1$ | $g_2$ |
|---|---|---|
| **A** | not normalized | replaced by $G_\alpha$ |
| **B** | not normalized | replaced by $G_\alpha^1$ |
| **C** | normalized | replaced by $G_\alpha$ |
| **D** | normalized | replaced by $G_\alpha^1$ |

As their performance level cannot be known beforehand, we had to test all four versions.

### Testing in a Frame-based Case

In a frame-based environment $g_1$ is usually defined simply as the sum of the frame-based phoneme costs. Thus, it is quite straightforward to apply some kind of operator on this level, only $g_1$ has to be changed from addition to this new operator. But to include this default function (addition), we have to use $G_\alpha^1$ or $G_\alpha^2$ again.

We can also apply a self-consistent mean operator as $g_2$, where we utilize it to construct hypothesis-level costs from the phoneme-level ones. Whichever version we applied as $g_1$, the result was already length-normalized; however, similar to the segment-based case, it may happen that in the final recognition, after applying $g_2$, a length-unnormalized version of $g_1$ works better. Thus we have the same four choices that we had previously; namely,

| **Variation** | $g_1$ | $g_2$ |
|---|---|---|
| **A** | not normalized | replaced by $G_\alpha$ |
| **B** | not normalized | replaced by $G_\alpha^1$ |
| **C** | normalized | replaced by $G_\alpha$ |
| **D** | normalized | replaced by $G_\alpha^1$ |

Testing all these variations would mean eight test cases (or even more if we apply $G_\alpha^2$ as $g_2$), and this would also not mean that we find the optimum by testing $g_1$ and $g_2$ separately. So, to simplify this matter, we found it logical to first test just the two variations for $g_1$; then we turn to $g_2$, using the $g_1$ operator which turned out to

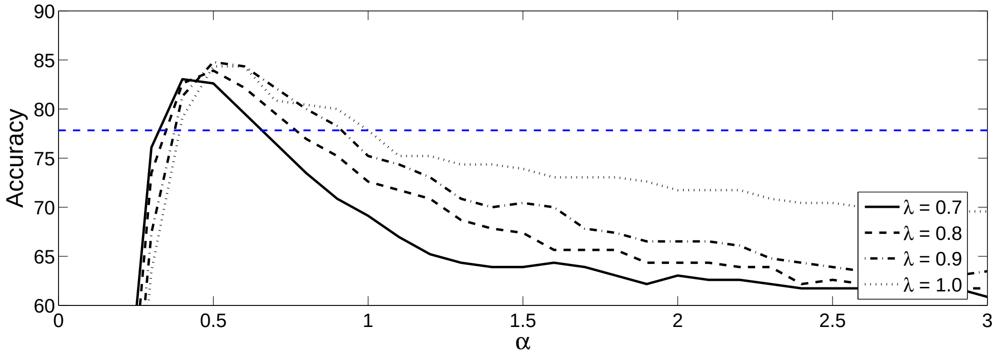

Figure 5.2: Recognition accuracy for the Children Database (isolated word recognition, segment-based context) with aggregation operator variation **D** ($\alpha \in [0.1, 3]$, $\lambda \in [0.7, 1.0]$) as $g_2$. The baseline value is represented by the horizontal dashed line.

be optimal. This means six tests overall, but note that one test can easily contain hundreds of test cases with different $\alpha$ values.

### 5.1.3 Results for the Children Database

We tested the behaviour of these operators in a segment-based context on the Children Database (see Section 2.5.3) in an isolated word recognition task. 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 [32], then the negative logarithmic value of its output was given to $g_2$. The baseline accuracy was $77.83\%$, which was a quite realistic value at the time of testing on this kind of database. The results of this experiment were published in [38].

We examined the above aggregation methods with $\alpha$ values ranging from $0.1$ to $3.0$ with a $0.1$ increments, with different $\lambda$s from $0.7$ to $1.0$. Although the step size used at $\alpha$ may seem too big for the first glance, this setup already means a total of $4 \times 30 \times 10 = 1,200$ test cases. In Figure 5.1 variations **A**-**D** can be seen with $\lambda = 1.0$. It can be seen that if we do not normalize $g_1$, the recognition will be relatively insensitive to changes in $\alpha$, which is a good property in applications as it means that the system is robust; on the other hand, type **D** achieved the best results. Surprisingly $\alpha = 1.0$ usually did not lead to the best results; rather the interval $[0.4, 0.7]$ seems the best for type **D** with the maximum value of $84.35\%$, and $[0.5, 2.0]$ for types **A** and **B**. The result was a relative error reduction of almost $30\%$.

In Figure 5.2 variation **D** can be seen with different $\lambda$ parameters. Although we expected that the decaying function controlled by the $\lambda$ value would be quite suitable for this kind of speech recognition environment, the results show that this was not the case. The best value for $\lambda = 0.9$ is slightly higher than that for $\lambda = 1.0$; otherwise the lower the value of $\lambda$, the worse the performance. So it seems that it is worth using the root-power mean operator as $g_2$, but not the decaying root-power mean operator, as it results in only a small improvement, but it is much more complex to apply and to tune.

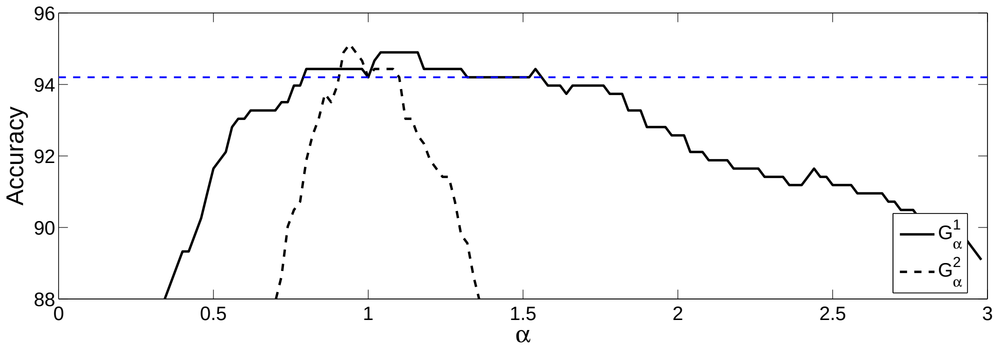

Figure 5.3: Recognition accuracy for the Telephone Database (isolated word recognition, frame-based context) with the $G^1_\alpha$ and $G^2_\alpha$ aggregation operators ($\alpha \in [0.02, 3]$) as $g_1$. The baseline value is represented by the horizontal dashed line.

### 5.1.4 Results for the Telephone Database

The behaviour of these aggregation operators in a frame-based environment was also tested on the Telephone Database [107] (see Section 2.5.3) with isolated word recognition on the 431 city names. The HTK system [111] was used as a reference and produced a score of $92.11\%$ on the same test data. We used Artificial Neural Networks for phoneme classification, with the standard 39 MFCC + $\Delta$ + $\Delta\Delta$ coefficients as features [50]. The minus logarithm of these scores were then aggregated into phoneme-level values by applying $g_1$ (addition by default), then the phoneme-level values were aggregated by $g_2$ (also addition by default). The baseline recognition accuracy of this system was $94.20\%$. The results of this experiment were published in [63].

In the first test we investigated the effect of the two variations of the root-power mean operator as $g_1$, with the $\alpha$ parameter ranging from $0.02$ to $3.00$ with a $0.02$ increment. Figure 5.3 shows the recognition results. It can be seen that if we use the variation $G^2_\alpha$ with $\alpha = 0.94$ instead of the default $\alpha = 1.00$, we can increase the recognition performance to $95.12\%$ (reducing the error level by $16\%$), thus we will use this setting later on. Almost the same level of improvement can be achieved with the other version (i.e. $G^1_\alpha$) with $\alpha = 1.06 \ldots 1.16$. Obviously the original $g_1 = G_\alpha(x_1, \ldots, x_j)$ version was also tested, but this test yielded very low recognition scores. Finally we applied the root-power mean operator as $g_2$ to construct hypothesis-level costs from the phoneme-level ones; Figure 5.4 shows the recognition figures for all four variations (i.e. **A**-**D**). Surprisingly, we were unable to improve the recognition accuracy this way: the optimal value was achieved by version **D** with $\alpha = 1.00$, which just means the default addition of the (length-normalized, phoneme-level) costs. This could be so because all the possible ways of improving the accuracy had already been exploited via $g_1$.

This test led us to conclude that it was worthwhile applying the root-power mean aggregation operator in this speech recognition context in a frame-based context too as it produced a $16\%$ reduction in the error rate; but the results suggest that it is enough to do it at the lower level (i.e. as $g_1$).

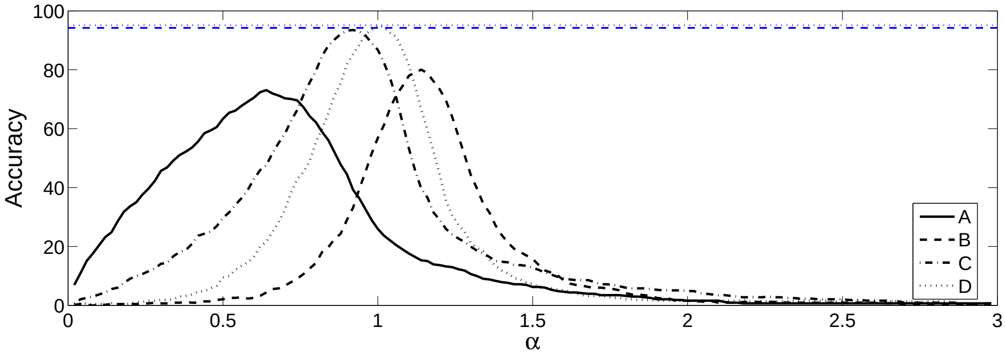

Figure 5.4: Recognition accuracy for the Telephone Database (isolated word recognition, frame-based context) using the four tested variations of the $G_\alpha$ aggregation operator ($\alpha \in [0.02, 3]$) as $g_2$. The baseline value is represented by the horizontal dashed line.

## 5.2 Using Standard Triangular Norms

The application of mean aggregation operators in the cost calculation problem of speech recognition turned out to be a good idea. The question which naturally arises now is what other kind of operators could be used (perhaps more efficiently, or mathematically better founded) for the same kind of problem. It is clear that practically any kind of operator can be applied either as $g_1$ or as $g_2$; however, only a small fraction of these operators can be even expected to work well in these cases. Also, it might be important to use the kind of operators which can be applied and fine-tuned easily.

To narrow the possible set of operators and – hopefully – exclude just a small fraction of the well-behaving ones, we set two criteria. One was that the behaviour of the operators should be easily modifiable, which is best done by operators with one or more parameters. The other was that since the default operator applied is the multiplication operator, it should be a "multiplication-like" operator. This latter criterion is indeed not a well-defined one, but its meaning is intuitively clear: the operators we apply should behave in a similar way to the multiplication operator. The *triangular norms* (or *t-norms*) are standard operators of fuzzy logic [24; 60], and they fulfil both requirements, which will be explained in detail later.

Another, and perhaps more convincing argument for the application of triangular norms comes from their role in fuzzy logic. They are used as the conjunction operator for values between 0 and 1, which can be easily interpreted as probability estimates. On the other hand, $g_1$ in a frame-based environment is used to calculate the probability of the whole given segment being a particular phoneme, provided that the first frame of this segment is this phoneme with the probability $x_1$ AND the second frame is the same phoneme with the probability $x_2$, and so on. Thus we have probability estimates for some events, and we want to calculate the probability of all these events happening at the same time. Similarly, $g_2$ (in both frame- and segment-based environments) is used to calculate the probability of a given hypothesis (phoneme- and segment-series) when its first segment corresponds to its first phoneme AND its second segment corresponds to its second phoneme, etc. This calls for the utilization of an operator having AND-like properties, just like the t-norms.

Note that this reasoning does not lead to the conclusion that fuzzy operators like these will eventually perform well in our actual problem. It just means that as the triangular norms have the sort of properties this problem requires, they might be a good model for this specific task. This issue can be settled only by testing them; but first we turn to the basic definitions of triangular norms, following the work of Fodor [28].

### 5.2.1 The Triangular Norms: Definitions

**Definition 5.3** *A function $T : [0,1]^2 \to [0,1]$ is a triangular norm (t-norm) if and only if it satisfies the following conditions:*

| | | | |
|---|---|---|---|
| *(T1)* | $T(1,x) = x$ | *for all* $x \in [0,1]$, | *(boundary condition)* |
| *(T2)* | $T(x,y) = T(y,x)$ | *for all* $x, y \in [0,1]$, | *(commutativity)* |
| *(T3)* | $T(x,y) \le T(u,v)$ | *for any* $0 \le x \le u \le 1$,<br>$0 \le y \le v \le 1$, | *(T is nondecreasing in both arguments)* |
| *(T4)* | $T(x,T(y,z)) = T(T(x,y),z)$ | *for all* $x, y, z \in [0,1]$. | *(associativity)* |

Note that (T1) and (T3) together imply that $T(0,x) = 0$ for all $x \in [0,1]$. It is easy to see that the product operator is also a t-norm, so we will refer to it as $T_P$ from now on. Next we need to make some more definitions.

**Definition 5.4** *A t-norm $T$ is said to be*

| | |
|---|---|
| *continuous* | *if $T$ as a function is continuous on $[0,1]$;* |
| *Archimedean* | *if $T(x,x) < x$ for all $x \in (0,1)$.* |

These two properties seem to be important for a function used either as $g_1$ or $g_2$. We should expect that a slightly different phoneme probability should affect the hypothesis probability by only a little bit as well; in other words, there are no sudden breaks between these kind of hypotheses, therefore $T$ should be continuous. On the other hand, if $T$ satisfies the Archimedean property, then the longer a word is, the closer the result (and hence the probability of the word pronounced) is to zero, which is also desirable. A similar line of reasoning holds for the $g_1$ case. Another reason for anticipating these properties is that our default operator ($T_P$) is also both continuous and Archimedean. Thus in the following we will use triangular norms which fulfil both these requirements. These two properties allow us to represent such triangular norms in a more general way:

**Theorem 5.1** *A t-norm $T$ is continuous and Archimedean if and only if there exists a strictly decreasing and continuous function $f : [0,1] \to [0,\infty]$ with $f(1) = 0$ such that*

$$T(x,y) = f^{(-1)}\big(f(x) + f(y)\big),$$

*where $f^{(-1)}$ is the pseudo-inverse of $f$ defined by*

$$f^{(-1)}(x) = \begin{cases} f^{-1}(x) & \textit{if } x \leq f(0) \\ 0 & \textit{otherwise.} \end{cases}$$

Moreover, this representation is unique up to a positive multiplicative constant. If the t-norm is strictly monotonously increasing (in which case we call the t-norm a *strict t-norm*), then $f$ is strictly monotonously decreasing, and the pseudo-inverse function $f^{(-1)}$ is the normal inverse $f^{-1}$. We will deal with this case, since in practical applications usually these kinds of operators are used as they are easier to handle. We say that a continuous Archimedean t-norm $T$ is *generated by $f$* if $T$ has such a representation; in this case $f$ is said to be an *additive generator* of $T$.

For the sake of completeness we also introduce the pairs of triangular norms called the *triangular conorms*. They play the role of the disjunction operator in fuzzy logic. They are very similar to the triangular norms, but they differ in the first axiom; i.e.

**Definition 5.5** *A triangular conorm (t-conorm) is a binary operation $S$ on the interval $[0,1]$, i.e. a function $S : [0,1]^2 \rightarrow [0,1]$, which, for all $x, y, z \in [0,1]$, satisfies $(T2) - (T4)$ and*

*(S1)* $S(x,0) = x$ *for all $x \in [0,1]$. (boundary condition)*

One such t-conorm is the addition operator, but its output can be no higher than 1 (i.e. $S(x,y) = \min(x+y,1)$). Similar to the case of t-norms, it can be shown that for any $x, y \in [0,1]$, $\max(x,y) \leq S(x,y) \leq 1$.

### 5.2.2 Using Triangular Norms in Speech Recognition

Like the mean aggregation operators, the above-defined functions can be used at both levels of probability calculation: either as $g_1$ where we construct phoneme-level phoneme probabilities from frame-level ones, or as $g_2$ where the phoneme-level scores are aggregated into hypothesis-level probability values. (Obviously the former one is available only if we are working in a frame-based context.) The first problem is that, for these tasks, we need operators which take any number of arguments, while the norms we have just introduced take only two. Fortunately though, this can be easily handled by using the associativity property (i.e. (T4)): for values $x_1, x_2, \ldots, x_n$ and the triangular norm $T$ we define the $n$-ary operator $T^n$ as $T^n(x_1, x_2, \ldots, x_n) = T(\ldots T(T(x_1, x_2), x_3), \ldots, x_n)$.

The second issue which needs to be settled is what else should be changed to use a fuzzy norm either as $g_1$ or $g_2$. (Remember that for the aggregation operators case we had to extend our definitions to include the basic operator, which was then the addition operator.) The answer is that, provided we apply the norms on the probability values and not on the costs, we need not change anything when utilizing them in this environment as they already act in a "multiplication-like" way; moreover, a number of standard operator families actually include $T_P$ as a special case. Their application is very straightforward either we perform segment-based or frame-based speech recognition, thus no such alterations are necessary like before when we tested the mean aggregation operators. Further, as the idea of not length-normalizing the result of $g_1$ (in a segment-based context) also resulted in a somewhat worse performance than doing it the other way, we see no reason why to bother with it this time.

Needless to say, the risk of underflowing is still present, so we still use the cost values instead of the probabilities, which means that we have to handle the costs as both the inputs and as the output of any t-norm used. Moreover, this should be done in such a way that we still avoid underflowing, thus a simple calculation of $e^{-p}$ on the incoming values, and the calculation of $-\log c$ on the result will probably not be enough. Fortunately it is not a very challenging programming task and can be handled quite easily.

The last open question is which norms should be tested. There could be very different answers to this question: try out as many standard norms as possible, test a general operator, or look for some other way. All these choices have their pros and cons, so we will go through them in this order.

### 5.2.3 The Standard Norms Used

Our first idea was to test a number of standard triangular norms. It was a plausible choice because there are many well-defined ones, and they are often recommended and have been successfully applied in a variety of problems, thus we may expect them to work well in our speech recognition environment. Next, we will introduce the triangular norms that are common in the fuzzy literature, and which have been tested here. One of the basic t-norms is the Lukasiewicz t-norm, which is defined as

$$T_L(x, y) = \max(x + y - 1, 0). \tag{5.6}$$

However, there also exist t-norm families, that is triangular norms with a parameter. Following Klement, Mesiar and Pap [60], we will list the common ones. Note that most of these t-norm families can have some parameter values which correspond to some basic t-norms, but we shall not mention them as they are of no interest to us here.

Schweizer-Sklar t-norms ($\lambda \in \mathbb{R}, \lambda \neq 0$):

$$T_\lambda^{SS}(x, y) = \Big(\max\left((x^\lambda + y^\lambda - 1), 0\right)\Big)^{1/\lambda} \tag{5.7}$$

Hamacher t-norms ($\lambda > 0$):

$$T_\lambda^{H}(x, y) = \frac{xy}{\lambda + (1 - \lambda)(x + y - xy)} \tag{5.8}$$

Yager t-norms ($\lambda > 0$):

$$T_\lambda^{Y}(x, y) = \max\left(1 - \left((1 - x)^\lambda + (1 - y)^\lambda\right)^{1/\lambda}, 0\right) \tag{5.9}$$

| | $T_\lambda^{SS}$ | $T_\lambda^{H}$ | $T_\lambda^{Y}$ | $T_\lambda^{D}$ | $T_\lambda^{SW}$ | $T_\lambda^{AA}$ | $T_\lambda^{MT}$ |
|---|---|---|---|---|---|---|---|
| continuous | $\lambda \in \mathbb{R}\backslash 0$ | $\lambda > 0$ | $\lambda > 0$ | $\lambda > 0$ | $\lambda > -1$ | $\lambda > 0$ | $0 \leq \lambda \leq 1$ |
| Archimedean | $\lambda \in \mathbb{R}\backslash 0$ | $\lambda > 0$ | $\lambda > 0$ | $\lambda \geq 0$ | $\lambda \geq -1$ | $\lambda \geq 0$ | $\lambda = 1$ |
| both | $\lambda \in \mathbb{R}\backslash 0$ | $\lambda > 0$ | $\lambda > 0$ | $\lambda > 0$ | $\lambda > -1$ | $\lambda > 0$ | $\lambda = 1$ |

Table 5.1: The intervals where the triangular norm families we deal with are continuous, Archimedean, and both continuous and Archimedean.

Dombi t-norms ($\lambda > 0$):

$$T_\lambda^D(x,y) = \frac{1}{1 + \left((\frac{1-x}{x})^\lambda + (\frac{1-y}{y})^\lambda\right)^{1/\lambda}} \tag{5.10}$$

Sugeno-Weber t-norms ($\lambda > -1$):

$$T_\lambda^{SW}(x,y) = \max\left(\frac{x+y-1+\lambda xy}{1+\lambda}, 0\right) \tag{5.11}$$

Aczél-Alsina t-norms ($\lambda > 0$):

$$T_\lambda^{AA}(x,y) = e^{-\left((-\log x)^\lambda + (-\log y)^\lambda\right)^{1/\lambda}} \tag{5.12}$$

Mayor-Torrens t-norms ($\lambda > 0$):

$$T_\lambda^{MT}(x,y) = \begin{cases} \max(x+y-\lambda, 0) & \text{if } \lambda \in ]0,1] \text{ and } (x,y) \in [0,\lambda]^2, \\ \min(x,y) & \text{otherwise.} \end{cases} \tag{5.13}$$

We should remark here that $T^{MT}$ with $\lambda = 1$ is the same as $T_L$. Recall that we intend to test t-norms which are both continuous and Archimedean. Hence it is important to determine the range of $\lambda$ values for each standard t-norm family where it fulfils both requirements. Table 5.1 lists these intervals.

The testing process is relatively simple for a given operator. First, the promising interval for $\lambda$ is determined via some preliminary tests; then a suitable step size is assigned to this interval. Next this region is explored with straightforwardly determined $\lambda$ values; this, in practice, meant hundreds of tests for each t-norm family in our case. Finally the $\lambda$ value with the best performance is chosen.

### 5.2.4 Results for the Children Database

First we tested the common triangular norms in a segment-based speech context on the Children Database (see Section 2.5.3). The feature set used was the MFCC-based one from the SUMMIT system [45]. Because the environment was a segment-based one, the triangular norms were applied as $g_2$, while $g_1$ remained the length-normalized output of the phoneme-identifying ANN. The task was isolated word recognition with

| | $T^{SS}_\lambda$ | $T^{H}_\lambda$ | $T^{Y}_\lambda$ | $T^{D}_\lambda$ | $T^{AA}_\lambda$ |
|---|---|---|---|---|---|
| interval | $[-3,1]$ | $]0,15]$ | $[180,320]$ | $]0,5]$ | $]0,20]$ |
| step | $0.01$ | $0.02$ | $0.5$ | $0.01$ | $0.05$ |

Table 5.2: The tested interval and the step sizes of the standard triangular norm families on the Children Database.

| t-norm family | $T_P$ | $T_L$ | $T^{SS}_\lambda$ | $T^{H}_\lambda$ |
|---|---|---|---|---|
| Best performance | $92.17\%$ | $0.26\%$ | $93.47\%$ | $92.60\%$ |
| Rel. error reduction | — | — | $16.60\%$ | $5.49\%$ |
| t-norm family | $T_P$ | $T^{Y}_\lambda$ | $T^{D}_\lambda$ | $T^{AA}_\lambda$ |
| Best performance | $92.17\%$ | $89.45\%$ | $93.47\%$ | $93.04\%$ |
| Rel. error reduction | $0.00\%$ | — | $16.60\%$ | $11.11\%$ |

Table 5.3: Recognition percentage scores and relative error reduction rates for the standard triangular norms applied on the Children Database (isolated word recognition, segment-based context); $T_P$ refers to multiplication (which is the baseline), while "—" denotes a failed test case.

the basic word recognition rate using $T_P$ of $92.17\%$; the *HTK* system [111] we used as a reference achieved a score of $92.60\%$ [41].

First we tested the Lukasiewicz t-norm ($T_L$), which behaved disastrously. The next step was to test the t-norm families that had a $\lambda$ parameter. For each t-norm family first the above-mentioned preliminary tests were carried out in order to determine the worthwhile region for its use. Unfortunately the Sugeno-Weber family did not produce any acceptable results even at this stage, so we excluded it from further tests. Table 5.2 shows the intervals we obtained and the appropriate step sizes. Next, the testing was actually performed on these the intervals and with these step sizes. The results can be seen in Figure 5.5, while the best performances of the t-norms tested are listed in Table 5.3. Note that $T_L = T^{MT}_1$, which could explain why the best performance of $T^{MT}$ turned out to be below $1\%$ by all measures. The reason for this is probably that the Lukasiewicz t-norm is a rather drastic one for probabilities appearing in a speech recognition environment: to get a result greater than zero, the sum of the two parameters must be over $1$. But the probability value of a hypothesis (one of these parameters) is usually close to zero, while the probability of the next phoneme (the other parameter) is rarely greater than $0.5$. Thus practically all hypotheses had a probability of zero, resulting in just a few hits which were lucky guesses. The bad performance of the Sugeno-Weber t-norm family can be similarly explained: the probability values generated were zero too often to get an acceptable result.

The remaining triangular norms had a good recognition performance, although the Yager t-norm family could not achieve the initial percentage value in this task. The others, however, matched or even outperformed the basic product operator: the Hamacher and the Aczél-Alsina family gave a slight improvement of $5.49\%$ and $11.11\%$ (in terms of the relative error reduction). The Schweizer-Sklar [93] and the Dombi [21]

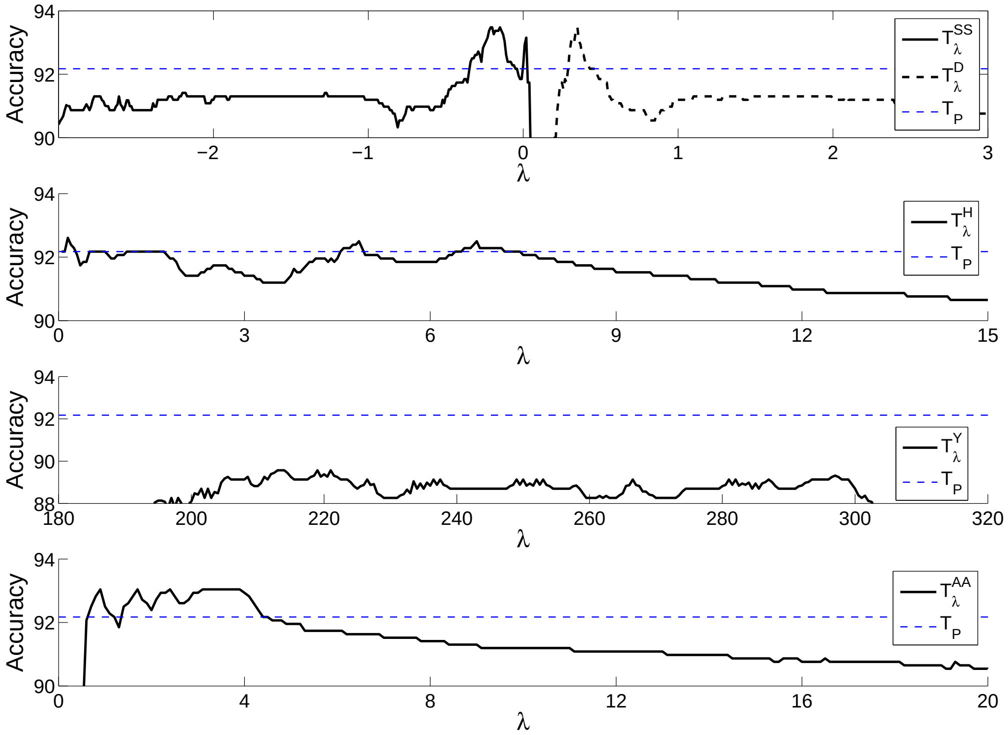

Figure 5.5: Recognition accuracy for the Children Database (isolated word recognition, segment-based case) using the Schweizer-Sklar, Dombi, Hamacher, Yager and Aczél-Alsina t-norm families as $g_2$, relative to the product operator.

t-norm families were the most effective for this task: we were able to achieve a $16.60\%$ relative error reduction using these two t-norms, causing the recognition percentage to increase from $92.17\%$ to $93.47\%$.

### 5.2.5 Results for the Medical Database

Another, perhaps more relevant experiment was made on the Medical Database (see Section 2.5.3), still in a segment-based context, and still using the features set of the SUMMIT system [45]. 150 sentences were taken from a medical context as the test database with a simple word 2-gram as the language model. Under these circumstances, the baseline values were $92.03\%$ and $91.69\%$ (correctness and accuracy, respectively) using the basic operator $T_P$. The results of this experiment were published in [41].

The testing process was similar to the one used for the case of the Children Database: first the regions where these standard t-norm families perform adequately were determined by preliminary tests. The Sugeno-Weber family did not produce any acceptable results, again, and now neither did the Yager family, so we omitted these as well, leaving only 4 triangular norm families. Table 5.4 shows the tested intervals and step sizes for these series of tests. The results can be seen on Figure 5.6, while the best performances are shown in Table 5.5.

The results are similar to those achieved on the Children Database, only the Yager t-norm family could not produce any acceptable results this time; on the isolated word recognition task, however, it also could not achieve the performance of the product

| | $T^{SS}_{\lambda}$ | $T^{H}_{\lambda}$ | $T^{Y}_{\lambda}$ | $T^{D}_{\lambda}$ | $T^{AA}_{\lambda}$ |
|---|---|---|---|---|---|
| interval | $[-2, 0.5]$ | $]0, 2]$ | — | $[0.1, 0.5]$ | $[0.9, 1.4]$ |
| step | $0.01$ | $0.01$ | — | $0.02$ | $0.02$ |

Table 5.4: The tested interval and the step sizes of the standard triangular norm families on the Medical Database.

| t-norm family | $T_P$ | $T_L$ | $T^{SS}_{\lambda}$ | $T^{H}_{\lambda}$ |
|---|---|---|---|---|
| Best accuracy | $91.69\%$ | $0.69\%$ | $92.61\%$ | $92.50\%$ |
| Relative error reduction | $0.00\%$ | — | $11.07\%$ | $9.74\%$ |
| Best correctness | $92.03\%$ | $0.81\%$ | $93.31\%$ | $93.19\%$ |
| Relative error reduction | $0.00\%$ | — | $16.06\%$ | $14.55\%$ |
| t-norm family | $T_P$ | $T^{Y}_{\lambda}$ | $T^{D}_{\lambda}$ | $T^{AA}_{\lambda}$ |
| Best accuracy | $91.69\%$ | — | $92.25\%$ | $92.03\%$ |
| Relative error reduction | $0.00\%$ | — | $6.73\%$ | $9.74\%$ |
| Best correctness | $92.03\%$ | — | $93.03\%$ | $92.73\%$ |
| Relative error reduction | $0.00\%$ | — | $12.54\%$ | $8.78\%$ |

Table 5.5: Accuracy and correctness values and relative error reduction rates with the standard triangular norms applied on the Medical Database (sentence recognition, segment-based context); $T_P$ refers to multiplication (which is the baseline), while "—" denotes a failed test case.

operator, so we do not regard it as a significant difference. As for the remaining four norm families, although the relative error reduction rates varied for the accuracy and correctness scores, there was a definite improvement in the values. This time again the Schweizer-Sklar [93] proved to be the best-performing triangular norm family with relative improvements of $11.07\%$ and $16.06\%$ (accuracy and correctness, respectively). The Hamacher family worked almost as well with rates of $9.74\%$ and $14.55\%$; moreover, the Dombi and Aczél-Alsina families also produced good results. Overall we may conclude that the application of triangular norms in the hypothesis probability aggregation problem was justified by the test results on both databases.

## 5.3 Using the Generalized Dombi Operator

The use of the standard triangular norms led to a nice improvement in performance, which showed that the application of triangular norms is indeed a good idea. But it is easy to see the major drawback of this approach: we cannot be sure when we have tested "enough" of the standard triangular norm families, as there exist so many of them, and perhaps we have not applied the most suitable one yet. However, using a more general operator family we could test several standard operators at a time, including a lot of triangular norms which are not among the standard norms.

Another advantage of using a more general operator is that it has more precise tunability properties; that is, a more general operator probably has more parameters,

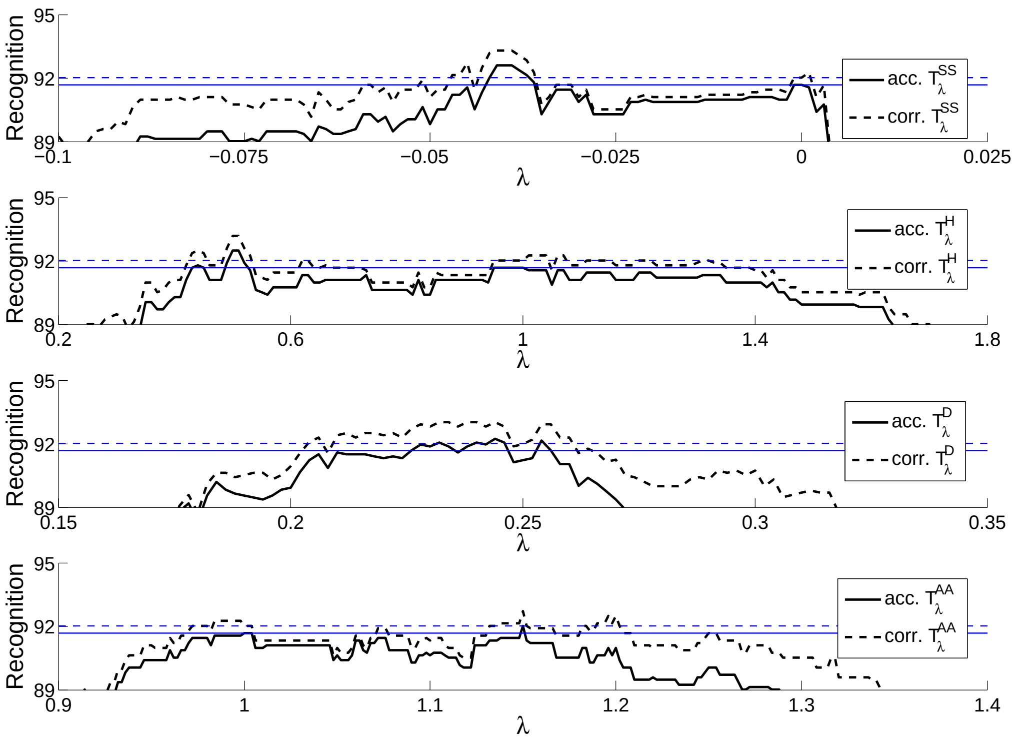

Figure 5.6: Recognition accuracy for the Medical Database (sentence recognition, segment-based context) using the Schweizer-Sklar, Hamacher, Yager, Dombi and Aczél-Alsina t-norm families, relative to the product operator. The sides of the tested regions where there was no information shown have been left out.

which means more opportunities to modify its behaviour in a hopefully better direction. But having more parameters also means a more difficult parameter setting process as the method used before cannot be realistically applied even for two parameters. Unfortunately there are practically no general operators like this available: the only one we could find is the Generalized Dombi Operator Family, so finally we applied it in our task.

## 5.3.1 The Generalized Dombi Operator Family

Dombi introduced a generalized family of triangular norms and their pairs called the triangular conorms [22]. Here we are interested in the triangular norm case, hence we will focus on them; in this case the operators can be represented in the following way:

$$T_{GD,\gamma}^{(\alpha)} = \frac{1}{1 + D_\gamma(x)} \qquad \alpha > 0 \tag{5.14}$$

where

$$D_\gamma(x) = \left( \frac{1}{\gamma} \left( \prod_{i=1}^{n} \left( 1 + \gamma \left( \frac{1 - x_i}{x_i} \right)^{\alpha} \right) - 1 \right) \right)^{1/\alpha} \tag{5.15}$$

and $\gamma > 0$. (The corresponding triangular conorm can be also derived from Eq. $(5.14)$ with $\alpha < 0$.) The generator function of the Generalized Dombi triangular norm is

| Type of Operator | Value of $\gamma$ | Value of $\alpha$ |
|---|---|---|
| Dombi | $\gamma = 0$ | $\alpha > 0$ |
| Hamacher | $0 < \gamma < \infty$ | $\alpha = 1$ |
| Einstein | $\gamma = 2$ | $\alpha = 1$ |
| Product | $\gamma = 1$ | $\alpha = 1$ |
| Drastic | $\gamma = \infty$ | $\alpha > 0$ |

Table 5.6: The well-known special cases of the Generalized Dombi Operator.

$$f_c(x) = \log\left(1 + \gamma\left(\frac{1-x}{x}\right)^{\alpha}\right), \tag{5.16}$$

with the same $\alpha$ and $\gamma$ values as before, i.e. $\alpha > 0$, $\gamma \geq 0$. (The generator function for the corresponding triangular conorm is the same, only with $\alpha < 0$.)

What makes this operator rather attractive and potentially useful is that it includes many well-known and widely-used triangular norm families as some special cases, among which some (like the Dombi or the Hamacher operator family) proved useful in our earlier tests [38]. Table 5.6 lists these operator types and the corresponding $\gamma$ and $\alpha$ values. But to apply this general operator in speech recognition first we have to find a way to set its parameters. Unfortunately, it is not as easy as it was for the standard triangular norm families, thus we need to describe it in more detail.

### 5.3.2 Tuning a 2-parameter Triangular Norm: Formulating a Maximization Problem

The optimization process is generally straightforward if only one parameter needs to be set. It was the case in our previous studies, where operators with one parameter were tested. Every triangular norm family has a certain range for a given task where it works satisfactorily for the given problem; all we need to do is to explore this interval with a sufficiently low step size. For triangular norm families with several parameters, however, this approach is not a practical one as it is unlikely that we can find the optimum by varying the parameters one at a time. With two parameters it is theoretically possible to explore a promising area with two nested cycles; for only $100$ and $100$ tested values (which are rather small), however, it already means $10,000$ test cases, which would take too much time to process. Thus we should find some other solution.

For this, let us denote the function which assigns the probability estimate to a hypothesis $h$ (consisting of a phoneme sequence and the corresponding segmentation bounds) by $G(h)$. Of course it involves using the $g_1$ operator and then the $g_2$ operator; hence this function in fact works with probability estimates of lower-level units. Let us concentrate on an utterance $u$; now we are interested in

$$h_{\max} = \arg\max_h G(h, u). \tag{5.17}$$

This formula means that we are looking for the finishing hypothesis $h_{\max}$ (i.e. one having a correct phoneme sequence and reaching the end of the utterance) which has the maximal probability estimate for the given utterance $u$. We have a priori knowledge about the actual transcript of $u$, and based on it, we can easily define a function which measures the accuracy of the hypothesis $h_{\max}$, i.e. $Acc(h_{\max}, u)$. It could be a 0/1-like function, or the accuracy or correctness score based on the inserted/deleted/replaced words, depending on the actual speech recognition task; but we expect that, for a correct hypothesis, it returns a value of $1$, and a lower value otherwise.

Next we will focus on $h_{\max}$. Since it is the finishing hypothesis with the maximal probability, it is easy to see that it strongly depends on the probability estimate calculation function $G(h, u)$. If we fix all possible parameters of the speech recognition system, $h_{\max}$ will *only* depend on $G$, and will be the hypothesis for which $G(u, h)$ is maximal. (In practice, of course, it will be the hypothesis with the maximal score $G$ *found by the search method*, but since the search process is also fixed, it does not affect our reasoning.) Thus we can express accuracy as a function of $G$ and $u$, i.e. $Acc(G, u)$.

In our special case, when we apply the Generalized Dombi Operator as $g_1$ and $T_P$ as $g_2$, it can be written in the form $Acc(G_{\alpha,\gamma}, u)$, which can be transcribed as $Acc\big((\alpha, \gamma), u\big)$. That is, the accuracy score for a given utterance $u$ will depend just on the parameters of the Generalized Dombi Operator, i.e. $\alpha$ and $\gamma$.

These functions can be easily generalized to a list of utterances: for an utterance vector $\mathcal{U} = (u_1, u_2, \ldots, u_N)$ and the hypothesis probability calculation function $G(h, u)$, the most probable finishing hypotheses will be $\mathcal{H} = (h^1_{\max}, h^2_{\max}, \ldots, h^N_{\max})$, respectively. The accuracy score can straightforwardly be extended to these hypotheses and utterances, i.e. $Acc\big((h^1_{\max}, h^2_{\max}, \ldots, h^N_{\max}), (u_1, u_2, \ldots, u_N)\big) = Acc(\mathcal{H},\mathcal{U})$. As the list of finishing hypotheses still depends only on the utterances and the probability estimate function $G(h, u)$, the accuracy function can be written as $Acc(G,\mathcal{U})$. In our case it takes the form $Acc(G_{\alpha,\gamma},\mathcal{U}) = Acc\big((\alpha, \gamma),\mathcal{U}\big)$; and, since we seek to maximize it for a given $\mathcal{U}$ (our actual test set), we are looking for the optimal pair of $\alpha$ and $\gamma$.

Having presented it formally, we still face the problem of finding the optimal pair of parameters for a triangular norm on our particular problem, so, at first glance, we seem to be no closer to the solution at all. But we can now consider it as simply optimizing a function with two parameters, which is a task with several solutions: we simply need to apply a global optimization method that looks for a global optimum, and apply it on the speech recognition process treated as a function of the parameters of the triangular norm used.

### 5.3.3 Using the Snobfit Package

For this optimization task we chose the Snobfit (Stable Noisy Optimization by Branch and FIT) [51] global optimization package. It is available as a Matlab 6 [74] package, and it is an optimization system designed especially for noisy functions which have parameters that vary between fixed bounds. The ranges of the $\alpha$ and $\gamma$ parameters were fixed by simple preliminary tests, and the function value to be optimized was obtained from the recognition accuracy value. Since Snobfit seeks to minimize this function

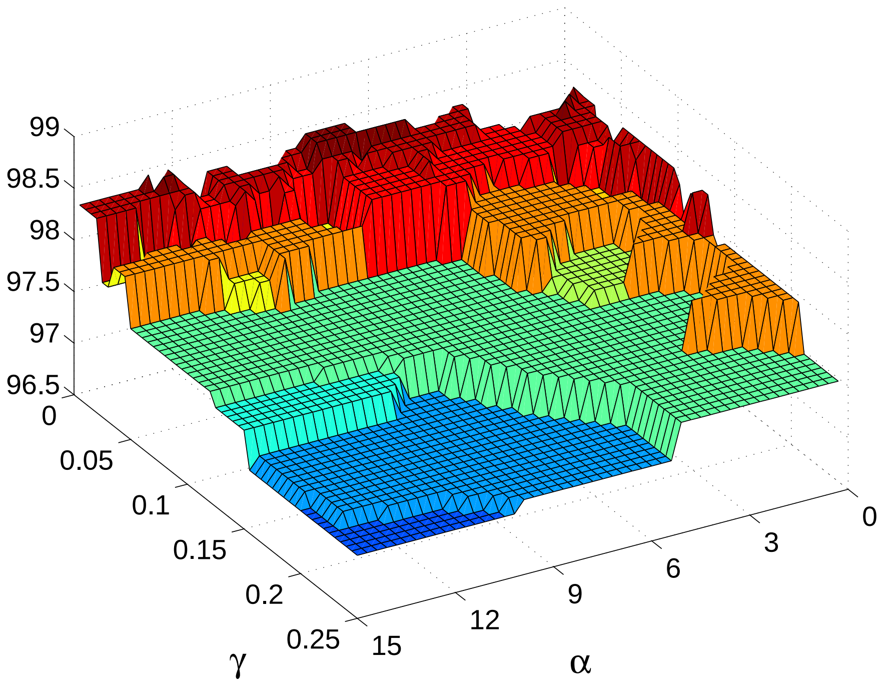

Figure 5.7: The accuracy values for the $\alpha$ and $\gamma$ parameters of the Generalized Dombi Operator on the Medical Database (sentence recognition, frame-based context).

value, we simply used the reciprocal rate of accuracy for this purpose. Another reason for choosing Snobfit is that the calculation of this function involves the execution of another application (i.e. our OASIS Speech Laboratory), and this operation is also supported, making the carrying-out of the optimization process much easier.

### 5.3.4 Results for the Medical Database

We used the Medical Database (see Section 2.5.3) for testing, on which sentence recognition was performed. We carried out our experiments in a frame-based context using the standard MFCC + $\Delta$ + $\Delta\Delta$ feature set [50], and replaced $g_1$ with the Generalized Dombi Operator, hence phoneme-level probabilities were calculated by it from frame-level ones. $g_2$ remained the basic product operator, based on the earlier test results where we were unable to gain any improvement at this level after optimizing $g_1$. The baseline values with $T_P$ as $g_1$ were $96.76\%$ and $98.38\%$ (accuracy and correctness, respectively). In addition, we calculated the correct number of whole sentences, which appeared to be $92.66\%$ (i.e. 139 correct sentences out of a total of 150). The results of this experiment were published in [36].

In Figure 5.7 and Figure 5.8 the test results, i.e. the recognition scores (accuracy and correctness, respectively) can be seen for the different $\alpha$ and $\gamma$ pairs. For the sake of clarity we show the resultant accuracy values on a three dimensional curve. Since the Snobfit algorithm performed tests on discrete $\alpha - \gamma$ pairs, and tested values which were promising, there could be huge areas where no test was made at all. For these points we used the accuracy value of the test case that was the closest.

The first and very interesting observation is that very few configurations led to a

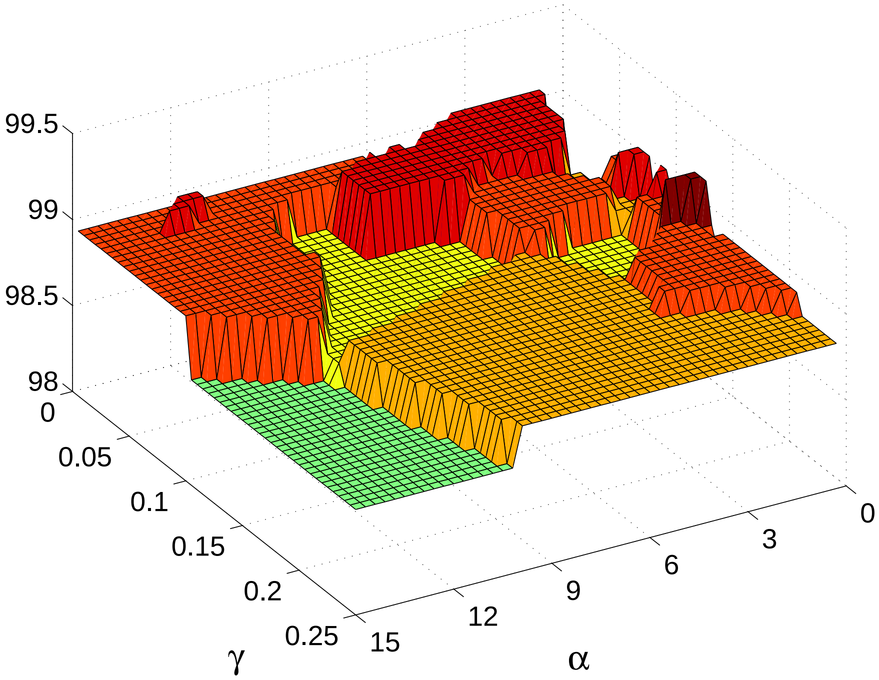

Figure 5.8: The correctness values for the $\alpha$ and $\gamma$ parameters of the Generalized Dombi Operator on the Medical Database (sentence recognition, frame-based context).

decrease in the accuracy score. The reason for this might simply be due to computer arithmetic (underflowing). The second observation is that the accuracy value improved quite significantly: from $96.76\%$ to $98.49\%$ when $\gamma = 0.1$ and $\alpha = 0.7$ and in the neighbourhood of this point. It actually corresponds to a relative error reduction of $53.40\%$, which is a surprisingly good result. The correctness value also increased from $98.38\%$ to $98.95\%$, which is a slightly lower, but nevertheless impressive relative error reduction of $35.19\%$. This difference is most likely due to the fact that we optimized the accuracy value, but since it is the more important of the two, it is best done this way. The percentage ratio of correct sentences also rose from $92.66\%$ to $94.66\%$, meaning a $27.25\%$ reduction in the sentence-level error value. These results support our earlier findings that it is worth applying triangular norms in the hypothesis probability calculation problem of speech recognition; moreover, they justified our idea of switching from the standard norms to a more general triangular norm (namely to the Generalized Dombi Operator).

## 5.4 More Flexible Norms: The Logarithmic Generator Function

Thus we found that switching from testing individual standard triangular norms to a more general operator could be quite beneficial as we could significantly improve the performance. Another step toward generality could be the replacement of the generator function of the triangular norm used. Recall that a strict, continuous and Archimedean triangular norm $T$ can be written in the form $T(x,y) = f^{-1}\big(f(x) + f(y)\big)$, where $f$ is the *additive generator* of $T$, and is a continuous, strictly decreasing function on the interval $[0,1]$; $f(0) = \infty$ and $f(1) = 0$. Moreover, for a given $T$, $f$ is unique up to a scalar. Thus the triangular norm applied can be also represented by its generator function, and if we could find a suitable way to model this function $f$, we could fine-tune its behaviour to suit our needs. Of course the performance of this method strongly depends on the approximation of the generator function, so we have to find a flexible yet simple representation. There are solutions available for this task [54], but they are more likely to be theory-oriented, while now we need an application-oriented approach. We will present such a solution, with begins by introducing the logarithmic generator function.

### 5.4.1 The Logarithmic Generator Function

To introduce the logarithmic generator function, we should first consider what changes the use of the generator function of a triangular norm introduces. In a typical evaluation we get a number of probability estimates as input $(p_1, p_2, \ldots, p_N)$, and a t-norm $T$; and we need to calculate

$$T^N(p_1, p_2, \ldots, p_N) = T(\ldots T(T(p_1, p_2), p_3), \ldots, p_N). \tag{5.18}$$

Now using the transcript $T(x,y) = f^{-1}(f(x) + f(y))$ we have that

$$T^N(p_1, p_2, \ldots, p_N) = f^{-1}(f(\ldots((f^{-1}(f(p_1) + f(p_2)), p_3), \ldots) + f(p_N)), \tag{5.19}$$

which can be readily rewritten as

$$T^N(p_1, p_2, \ldots, p_N) = f^{-1}\Big(\sum_i f(p_i)\Big). \tag{5.20}$$

Now recall that in a typical speech recognition environment, to avoid underflowing, instead of a probability $p$ we use $c = -\log p$. It would be handy if we did not have to bother with converting cost to probability and vice versa all the time, but the triangular norms, unfortunately, work only on probabilities. The first step to overcome this difficulty is to incorporate this conversion into Eq. 5.20, i.e.

$$-\log\Big(f^{-1}\Big(\sum_{i=1}^{N} f(e^{-c_i})\Big)\Big). \tag{5.21}$$

As we always apply this formula, it is straightforward to include the calculation of the negative exponential into $f$, so we will define the logarithmic generator function as

$$\phi(x) = f(e^{-x}). \tag{5.22}$$

Now we can write

$$\phi^{-1}\Big(\sum_{i=1}^{N}\phi(c_i)\Big) = -\log\Big(f^{-1}\Big(\sum_{i=1}^{N}f(e^{-c_i})\Big)\Big) \tag{5.23}$$

$$= -\log T(e^{-c_1}, e^{-c_2}, \ldots, e^{-c_N}), \tag{5.24}$$

so using the logarithmic generator function $\phi(x)$ in exactly the same way as we did with the additive generator function $f(x)$ will lead to the calculation of the same triangular norm $T$, only with the corresponding cost values instead of the probabilities both as arguments and as the result. As $f(x) : [0,1] \to [0,\infty)$ was a strictly decreasing function with $f(1) = 0$, the logarithmic generator function $\phi(x) : [0,\infty] \to [0,\infty]$ is strictly increasing, and $\phi(0) = 0$. As the additive generator function was unique up to a multiplicative constant, the same is true for the logarithmic generator function.

### 5.4.2 Modelling the Logarithmic Generator Function

Now we will turn to modelling this logarithmic generator function. Almost any approximation could be used to estimate the logarithmic generator function; we chose to model it with a piecewise linear one for two reasons. First, it is quite simple to handle: both $\phi$ and $\phi^{-1}$ can be implemented very easily. Secondly, it is a very flexible representation: the family of all strict t-norms with a piecewise linear logarithmic generator $\phi : [0,\infty] \to [0,\infty]$ with finitely many breakpoints, such that $\lim_{x\to\infty}\phi'(x) = 1$, is dense in the family of all strict t-norms with respect to the topology of uniform convergence. A proof of this is just a standard compactness argument.

Henceforth let $\phi = \phi^{m_1,\ldots,m_n}_{a_1,\ldots,a_n} : [0,\infty] \to [0,\infty]$ be the piecewise linear, strictly increasing function with break points on the domain as $0 = a_0 < a_1 < a_2, \ldots, a_n < a_{n+1} = \infty$ and with steepness values $m_1, m_2, \ldots, m_n > 0$ respectively and $m_{n+1} = \lim_{x\to\infty}\phi'(x) = 1$. That is,

$$\phi(x) = (x - a_j)m_{j+1} + \sum_{i=1}^{j}(a_j - a_{j-1})m_j, \qquad a_j \le x < a_{j+1}. \tag{5.25}$$

This representation actually has several advantages. If the $a_i$ control points are fixed, a function $\phi$ can be described by the vector of the $n$ steepness values, making it easy to optimize. On the other hand, the function $\phi$ is unique up to a positive multiplicative constant; now, by setting $m_{n+1}$ to 1, we fix exactly one of these equivalent representations. We have the freedom to position the control points at values where they represent the problem we are currently modelling as accurately as possible.

This way, by keeping all the $a_j$ values and every other possible setting of the speech recognition environment fixed, this problem can be simplified to that of a maximization task in an $n$-dimensional space, in a similar way to the Generalized Dombi Operator case. That is, for a vector $\overline{m} = (m_1, m_2, \ldots, m_n)$ and the utterances $\mathcal{U} = (u_1, u_2, \ldots, u_N)$, we maximize the accuracy as a function of $\overline{m}$ on this utterance set (i.e. $Acc(\overline{m},\mathcal{U})$).

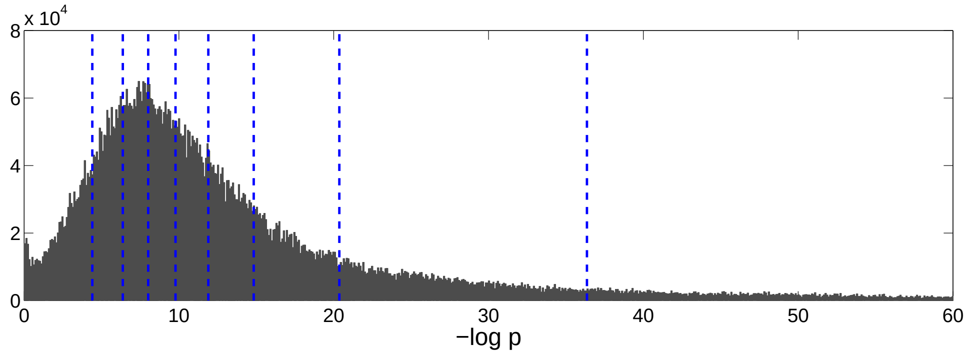

Figure 5.9: A histogram of the $-\log p$ values appearing during a typical speech recognition process using multiplication, on the interval $[0, 60]$, and the suggested positioning of $n = 8$ control points.

### 5.4.3 The Choice of Control Points

The only remaining task is to accurately position the control points. For this we introduce a method based on a simple test: let us perform an ordinary speech recognition process with the default operator, i.e. with $T(x, y) = xy$, $f(x) = -\log x$. During this test let us note which $x$ and $y$ values are passed to the operator (and thus to the generator function $f$). Owing to the commutativity property we do not need to distinguish between the two arguments, i.e. $x$ and $y$. Next, calculate a histogram of the $-\log$ of recorded values, i.e. for each value note how many times it appears. Finally, to assign the $n$ control points, divide this histogram into $n+1$ equal-sized parts: the control points will be the borders between these regions. This way, during a typical run, roughly the same number of evaluations will fall into each region of the function $\phi$ between two adjacent control points, so that each steepness value will have about the same importance.

The actual function $f$ (and thus, the triangular norm $T$) can be easily calculated from $\phi$. Of course it will not be piecewise linear, but a piecewise exponential function with $n+1$ negative exponents. It will be continuous, but not smooth, i.e. its derivative will be a discontinuous function (except for the case where each $m_i$ steepness value equals $1$, which is just the product case).

### 5.4.4 Testing

At this point it is quite easy to test a triangular norm with a piecewise linear logarithmic generator function in speech recognition either as $g_1$ or $g_2$. First the control points of the logarithmic generator function have to be set; we used the method suggested above and positioned these by running a standard speech recognition test using multiplication, i.e. $f$ is $-\log x$. The resulting histogram of the actual $-\log x$ values and a sample list of control points can be seen in Figure 5.9, while some possible piecewise linear logarithmic generator $\phi$ functions are shown in Figure 5.10.

The points were then positioned at the values between $n+1$ equal-sized regions. Needless to say, if we attempt to change both $g_1$ and $g_2$ at the same time, we will end

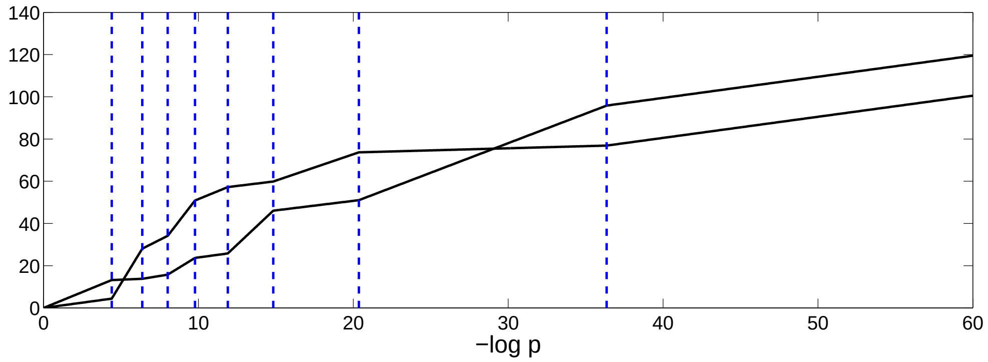

Figure 5.10: Two logarithmic generator functions with control points obtained from Figure 5.9.

up having twice as many control points, thus twice as many steepness values, resulting in an optimization problem with twice as many dimensions as we would have otherwise. There is nothing wrong with this in theory, since an optimization method may handle it quite well, but we performed experiments by changing just one operator at a time as the results of the previous tests indicate that there is no point in changing $g_2$ if $g_1$ had already been optimized. Moreover, if we increase the number of dimensions, we have to keep in mind *curse of dimensionality* [8].

Lastly just the choice of global optimization method is left; we chose the Snobfit package for the same reasons as we did for the Generalized Dombi Operator case.

### 5.4.5 Results for the Medical Database

The testing environment was practically identical to that of the Generalized Dombi Operator: we used a frame-based setting on the Medical Database on a sentence recognition task (see Section 2.5.3), with the standard 39 MFCC features [50]. The baseline values were also the same: $96.76\%$ and $98.38\%$ (accuracy and correctness, respectively), with the number of correctly recognized whole sentences being $92.66\%$ (i.e. 139 correct sentences out of a total of 150). The results of this experiment were published in [43].

We carried out experiments with $n = 8$ and $n = 16$ control points. As $\phi$ is a strictly monotonously increasing function, we can say that $m_j > 0$. On the other hand, there is no definite upper bound; but since $m_{n+1} = 1$, we thought that $m_j \leq 10$ would be sufficient. Thus we looked for a point in an $n$-dimensional hypercube, namely $(0, 10]^n$, which results in a maximal accuracy value.

Table 5.7 gives the best performances of the methods used. Besides the baseline values of the product operator and the results of the new modelling approach, the performance of two other t-norms are shown for reference: that of the Dombi triangular norm family, and that of the generalized Dombi operator family [22], which were obtained from earlier tests.

As can be seen, beyond the baseline scores (where the product operator was used) significant improvements could be attained. Even by using the somewhat "simple"

| Method | Accuracy | Correctness | Sentences |
|---|---|---|---|
| Product (baseline) | $96.76\%$ | $98.38\%$ | $92.66\%$ |
| Dombi t-norm | $97.57\%$ | $98.84\%$ | $93.33\%$ |
| Generalized Dombi operator | $98.49\%$ | $98.95\%$ | $94.66\%$ |
| Modelled t-norm, $n = 8$ | $98.27\%$ | $98.84\%$ | $94.00\%$ |
| Modelled t-norm, $n = 16$ | $98.84\%$ | $99.19\%$ | $96.00\%$ |

Table 5.7: The best accuracy and correctness values obtained for the tested triangular norms on the Medical Database (sentence recognition, frame-based context), and the ratio of correct sentences.

Dombi operator, the error rates can be reduced quite significantly. Using the generalized Dombi operator leads to even better results, but its application is more complicated because it has two parameters instead of just one. However, with the logarithmic generator function no further difficulties arise, and it can lead to a better performance (i.e. higher accuracy and correctness scores). In addition, the ratio of correct sentences can be increased. To get this result, however, there should be a sufficient number of control points in order to provide the t-norm with enough freedom to fit the problem it is applied to. This might be the reason why our proposed method with $n = 8$ could not attain the performance of the generalized Dombi operator. (It still performed much better than the product case, however, especially for the accuracy score, which it was optimized for.)

But with $n = 16$, it was able to outperform all the methods tested in all measurements: it led to relative error reduction scores of $64.20\%$ and $50.00\%$, accuracy and correctness, respectivel. Quite clearly, when we have enough freedom to optimize the generator function (and thus, the behaviour of the triangular norm it generates), it can fit the particular task even better than the well-performing classical t-norm families we tested. The use of too many control points, however, may lead to a case where the search space is so high dimensional that finding an optimum can be hard or almost impossible. Fortunately with $n = 16$ this was not the case, so this choice of $n$ seems to be a good compromise between easy optimization and robustness in our case.

We would like to stress that the use of the logarithmic generator function for modelling t-norms is by no means limited to the field of speech recognition. Although our tests were restricted to this field, we see no reason why this technique should not work in any area that makes use of triangular norms. Its application may, of course, require some small modifications to find the right value of $n$.

## 5.5 Summary

Following the decomposition of the speech recognition problem described in Chapter 2, there are two places where we could apply various kinds of operators: both are in the hypothesis probability/cost calculation problem, but at a different level. The first one is when we aggregate phoneme-level probabilities from lower-level information sources (denoted by $g_1$), while the second one is when we construct a hypothesis-level score from the phoneme-level values (denoted by $g_2$). Evidently, neither level is straightforward in every speech recognition system; usually $g_1$ does not involve the use of an operator in a segment-based framework, and it can also be quite hard to incorporate an operator in a HMM-based speech recognition system as $g_2$. This chapter concentrated on employing different aggregation operators at these levels in order to reduce the error rates. It covers theses II/1 and II/2, and is covered by publications [36; 38; 40; 41; 43; 62; 63].

First mean aggregation operators were applied, namely the root-power mean operator and its introduced variation, the decaying root-power mean operator (which makes earlier values less important) were tested on the Children Database and on the Telephone Database. The root-power mean operator yielded an improvement of almost $30\%$ on the Children Database and an improvement of about $16\%$ on the Telephone Database, meaning that its application can help the recognition process quite significantly with practically no loss in speed. (These tests were those of isolated word recognition, thus these scores refer to the decrease in the word error rate.) Another result was that if we apply an operator at the lower level (when we aggregate phoneme-level cost values from frame-level ones, i.e. as $g_1$), there is no point in doing this at the higher one (when we calculate hypothesis-level cost values from the phoneme-level ones, i.e. as $g_2$) as it yields no further improvement. Using the decaying root-power mean operator, however, only led to an insignificant improvement in the accuracy scores, which was an unexpected result as this operator weights earlier observations as less important, which characteristics seemed to fit to our particular problem.

As the application of the mean aggregation operators was not that easy, we looked for other operators to test. We set some criteria about the kind of operators to apply, namely they should be as close in behaviour to the original applied operator (i.e. multiplication) as possible, and further, they should be easily tunable by having (one or more) parameters. This led us to the utilization of standard triangular norm families taken from fuzzy logic theory, which fulfil both criteria: they have multiplication-like properties (like having unit 1, sink 0, being associative and commutative), and usually they have a $\lambda$ parameter which controls their behaviour. Due to their similarity with the product operator their application was much more straightforward than it was for the mean aggregation operators. They were applied in a segment-based context, hence they could be applied only as $g_2$. Tested on the Children Database in an isolated word recognition task, some of the norms (namely the Schweizer-Sklar and the Dombi families) could be configured so as to achieve a $16.60\%$ relative reduction in the error score. These standard norms were also tested on the Medical Database in a sentence recognition task, where RER scores of $9-11\%$ and $8-16\%$ could be achieved (accuracy and correctness, respectively), thus their use also led to a significant improvement in the recognition scores. (Note that these values, obtained in a segment-based context, cannot be directly compared to the other test results in this chapter.)

Next we looked for a more general operator to apply, and we found the General Dombi Operator a suitable choice. This triangular norm has all the kind of properties we required earlier, but it also has the potential of more precise tuning as it includes many standard triangular norms as special cases, and supplies a smooth transition between them via its two adjustable parameters ($\alpha$ and $\gamma$). However, it also required special treatment as setting two parameters is much harder than doing it with just one. To address this task, we turned the speech recognition problem into an optimization one and applied a standard optimization package to find a local optimum point. With the parameter values found this way on the Medical Database (in a sentence recognition task), we were able to reduce the error rates by a surprisingly large amount: the achieved relative error reduction scores of $53.40\%$ and $35.19\%$ (accuracy and correctness, respectively) suggest that it was indeed worth applying such a general triangular norm for this task.

Finally an attempt was made to see if the triangular norms could be modelled in a more general way. To do this we concentrated on the *additive generator* of the triangular norms, for which an application-oriented estimation method was introduced. For it we introduced the *logarithmic generator*, which makes the given t-norm work directly on the cost values (i.e. $-\log p$) instead of the probability values. This logarithmic generator was estimated piecewise linearly, keeping the steepness values constant between certain intervals. The bounds of these intervals (the *control points*) were positioned using the result of a histogram in such a way that roughly the same number of uses fell into each interval, making each steepness value important at the same level. After arranging the steepness values in the form of a vector $\overline{m}$, the whole speech recognition problem could be treated as an optimization task for $\overline{m}$, maximizing the recognition accuracy. The results of the experiments confirm that it is possible to reduce the error rates this way even more than it was possible via the Generalized Dombi Operator, leading to relative error reduction scores of $64.20\%$ and $50.00\%$ for accuracy and correctness, respectively. Moreover, this method could also be applied in other areas as well where fuzzy norms are used, thus its usage is not limited to speech recognition.

Finally it should be pointed out that the actual best parameter values are not the key issue, because they may vary in different settings (the feature set, frame or segment-based model applied, database used, etc.). The actual value of the parameter or parameters (i.e. $\lambda$, $\alpha$ and $\lambda$, the $\overline{m}$ vector of the steepness values) can be tuned to suit the particular problem. What we think important is the ability of an operator to actually improve the recognition scores. In particular, it is an improvement which requires practically no additional running time since in speech recognition the signal processing, phoneme-identifying and searching tasks are so CPU demanding that they make the calculation of a few operators practically negligible.

# Chapter 6

# The Anti-Phoneme Problem

*"But what about the End of the Universe? We'll miss the big moment."*
*"I've seen it. It's rubbish," said Zaphod, "nothing but a gnab gib."*
*"A what?"*
*"Opposite of a big bang. Come on, let's get zappy."*
*Douglas Adams – The Restaurant at the End of the Universe*

In the segment-based approach of speech recognition [101] the problem of determining whether speech segments correspond to a real phoneme or not arises quite frequently. To solve it, we have a large number of examples for such correct segments in the form of a hand-labelled corpus; on the other hand, there is no sure way of having counter-examples (or *outliers*) since these should contain anything which could occur in a sound recording that is not a correct phoneme (various kinds of noise, segments longer or shorter than one phoneme, etc.). These incorrect phonemes are the "anti-phonemes" [31; 68; 101], which led researchers to call the whole problem the "anti-phoneme problem" (although perhaps it would be more logical to call these incorrect phonemes "aphones"). It is quite easy to see that any solution applied to this task could affect the accuracy of a (segment-based) speech recognition system.

In contrast with the conventional classification problem of Artificial Intelligence, where there are examples from other classes, in this case just examples of the phoneme class are given. This task is called *one-class classification*, and it seeks to characterize this one given class as accurately as possible, to distinguish it from anything else [14; 67; 96].

To handle this task we have two basic choices: we can choose a tool which models just the positive examples, or we can somehow generate counter-examples for the given positive ones and then train some machine-learning method to separate these two classes. This counter-example generation can also be done in two different ways: in a task-specified manner (i.e. in a way that is specific to the given positive examples), or by applying a general counter-example generation algorithm not specially designed for this particular problem. So, in the end, we have three choices, all which of course can be implemented differently: different methods can be used to model all the positive examples together, to generate the counter-examples, or to separate the classes of examples and counter-examples. There are so many variations that they cannot all be tested, but we would like to check a representative solution from all three approaches.

In this chapter first we briefly describe the anti-phoneme detection problem. Next, we describe the methods we applied for this task and explain how we tested the new methods. Finally we will present our results and draw our conclusions.

## 6.1 The Anti-Phoneme Problem

Recall (see Chapter 2) that in the speech recognition problem we are looking for the most probable word $\hat{w} \in W$ such that

$$\hat{w} = \arg\max_{w \in W} P(A|w)P(w). \tag{6.1}$$

Concentrating on $P(A|w)$, we assume that the $A$ signal can be divided into a sequence of non-overlapping segments, and each of these segments corresponds to one of the $o_j$ phonemes of the word $w = o_1, \ldots, o_n$. As the correct segmentation of the signal is not known, it appears as a hidden variable $S$ in the segment-based formulation, the proper value of which has to be found. That is,

$$P(A|w) = \arg\max_{s \in S} P(A, s|w). \tag{6.2}$$

There are several ways of decomposing $P(A, s|w)$ further, depending on our modelling assumptions. What is common to all the derivations is that they trace the global probability back to the probabilities associated with the segments. The segments are usually assumed to be independent, and hence the corresponding local probability values will simply be multiplied. For example, Glass et al. employ the formula [31]

$$P(A, s|w) = \prod_{j=1}^{n} \frac{P(A_j|o_j)}{P(A_j|\alpha)} P(s_j|o_j), \tag{6.3}$$

where $P(s_j|o_j)$ is a duration model, $A_j$ denotes the acoustic feature set extracted from the $j$th segment, while $\alpha$ denotes the "anti-phoneme": a unit that covers all the possible signal samples that do not correspond to a real phoneme. Tóth et al. propose the formula [100]

$$P(A, s|w) = \prod_{j=1}^{n} \frac{P(o_j|A_j)P(\overline{\alpha}|A_j)}{P(o_j)}, \tag{6.4}$$

where $P(\overline{\alpha}|A_j)$ denotes the probability that the given segment is *not* an anti-phoneme.

The main difference between the two models is that in the former the acoustic observations are conditioned on the class labels (or the anti-phone), while in the latter it is the other way round. Hence, in practice the components of the first formula are conveniently modelled by generative techniques, while discriminative ones are more straightforward for the latter. Nevertheless, the posterior and class-conditional probabilities can always be easily converted to each other by applying Bayes' formula, so the formal derivations do not limit our choice of the machine-learning algorithm to be applied. As the formula in Eq. (6.4) is used by default in our OASIS framework, it is straightforward to apply this one during our experiments. We think, however, that this choice does not affect our tests performed.

In this chapter we will focus on the modelling of the anti-phoneme component $P(A_j|\alpha)$ or $P(\overline{\alpha}|A_j)$. It can be seen from Eq. (6.4) that these values affect the overall probability of a hypothesis, thus, similar to the application of operators, we can change the most probable hypothesis to another one, by which we can modify the recognition accuracy. Obviously, the two classes $\alpha$ and $\overline{\alpha}$ are the complements of each other, so the posterior-based formulation suggests that we should apply a two-class machine-learning algorithm to separate them, but as we have no training examples for the anti-phoneme class $\alpha$, a one-class type of modelling of $P(A_j|\alpha)$ seems more appealing.

## 6.2 Handling the Anti-Phoneme Problem

So we have plenty of examples for real phonemes belonging to various classes (including that of silence), and we want to somehow characterize them to distinguish them from any other speech segments we might encounter. Essentially, there are two approaches for solving a one-class classification problem like this. First, we could find a method which can by itself model all the examples which appear. The second approach is to take the actual occurrences of phonemes as positive examples, somehow generate "incorrect phonemes" as negative ones, and separate these two classes by some statistical classification method like Artificial Neural Networks (ANNs) [10] or Support Vector Machines (SVM) [90; 97] (after the feature-extraction part, where every phoneme is transformed into a fixed-length, $d$-dimensional vector, for which we use the feature set already employed for phoneme identification). Naturally we do not have actual training examples for the anti-phonemes as we do for phonemes; thus, the course of this automatic generation is by no means straightforward.

In this chapter we will describe one solution for one-class modelling, and two approaches for automatic counter-example generation. From the latter two the first one is a speech recognition-specific method [100], while the second counter-example generator tool is the specific use of a general-purpose algorithm [6]. Along with the utilization of the latter, we also propose some modifications to it which do not really affect its results, but have a surprisingly good effect on its running speed.

### 6.2.1 Modelling All Phonemes with Gaussians

Perhaps the most straightforward idea for describing all phonemes is that of converting their occurrences to a probability distribution over the feature space, and then modelling them with the sum of some Gaussian curves. And this is essentially what Gaussian Mixture Models (GMMs) [25] do: for a correct phoneme first the feature extraction part is done, then a clustering is performed on the set of the resultant $d$-dimensional

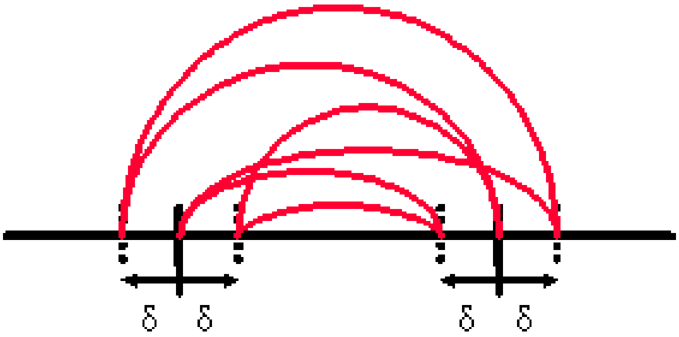

Figure 6.1: The mechanism of the Incorrect Segment Sampling Algorithm.

vectors (the positive examples) in order to divide them into $n$ distinct subsets. Finally one $d$-dimensional Gaussian is placed over every subset by calculating the mean and variance values of its elements [32].

We included this method here for three reasons. First, it is a good example for modelling all the phonemes (just the positive examples). Second, this solution usually works quite well. And third, this is the usual way for solving the anti-phoneme problem [14; 31], so its performance can be viewed as a kind of a baseline.

### 6.2.2 Incorrect Segment Sampling Algorithm

Tóth introduced a method for generating "incorrect" segments [100]. It is based on the fact that the negative examples are probably parts of speech with incorrect segmentation bounds; i.e. they commence and/or end at positions where there is no real bound between phonemes. If we know the real phonetic boundaries – and for any training database this is the case –, then it is quite easy to generate negative examples by choosing incorrect phoneme boundaries for the start and/or end segment bound.

The actual solution is quite straightforward: for each phoneme several anti-phonemes are generated by placing one or both phoneme boundaries earlier or later by $\delta$ milliseconds. This can be done in eight possible ways (both bounds can be earlier, at the original place or later; from the resulting nine variations one is the correct phoneme, thus the remaining eight could be used as counter-examples), from which six are selected as actual counter-examples (see Fig. 6.1). Note that in this method the feature extraction is done *after* the counter-examples have been chosen. This is the other solution which can be found in the literature.

### 6.2.3 Using General Counter-example Generation

Rather than applying a speech recognition-specific method, we can use a general counter-example generation algorithm. Thus, we will apply a method which takes all our examples (all the correct phonemes *after the feature extraction part*) as the elements of one class, and generates a number of counter-examples. Its input will be a

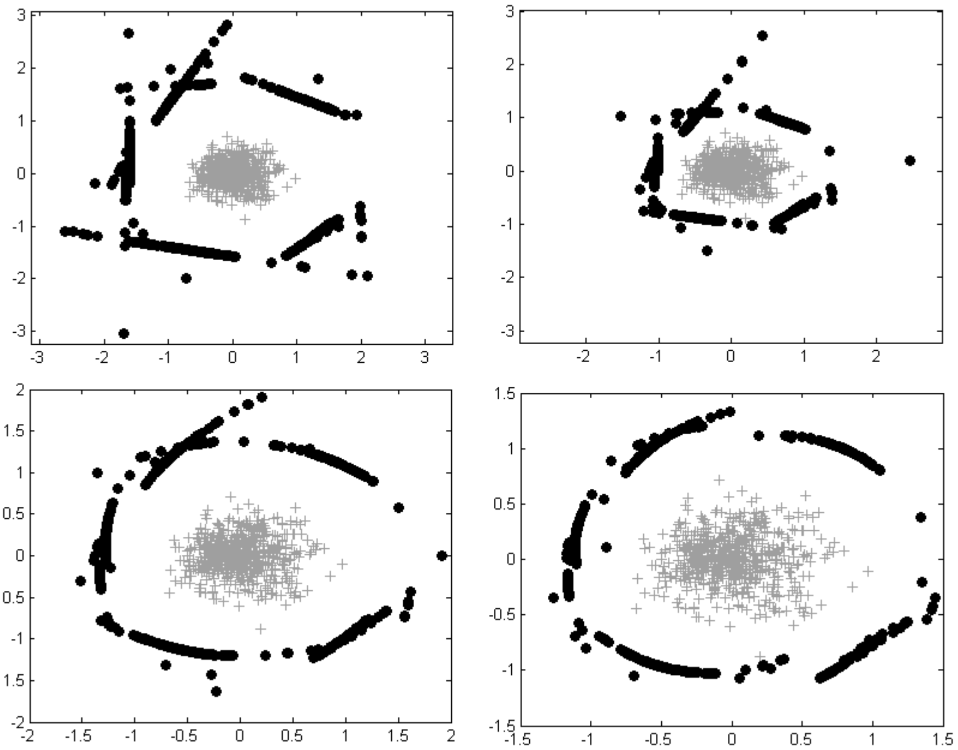

Figure 6.2: These figures show the generated counter-examples with different settings. Left to right: 1st: $(dist, curv) = (1, 0)$, 2nd: $(dist, curv) = (0.4, 0)$, 3rd: $(dist, curv) = (1, 0.5)$, 4th: $(dist, curv) = (1, 1)$. One can see that $curv = 0$ will generate points of a hyperplane, while $curv > 0$ will generate points of a better fitting boundary hyper-surface. With the $dist$ parameter, the distance between the boundary points and the generated counter-examples can be controlled.

set of $d$-dimensional vectors ($X$, $|X| = N$), the positive examples, and its output will be another set of $d$-dimensional vectors, which somehow represent the opposite of our examples. If this method works well, then, by examining these two sets, the nature of the positive examples can be determined. Evidently this process relies heavily on the set of features extracted, but it is natural to assume that a good counter-example generator algorithm will work well on the feature set already used for phoneme classification.

Bánhalmi introduced one such counter-example generation algorithm, which we will now apply for the anti-phoneme task. First we will briefly describe it; for more details, see [6]. Note that in speech recognition usually huge databases are used compared to other common areas of Artificial Intelligence as it is usual to have over a hundred thousand phonemes for training. This of course affected the way this algorithm was used here: when we had the opportunity to choose from different solutions for a subtask, we always opted for the faster one. For example an SVM-based boundary point detection algorithm was suggested, but a $min/max$ based, approximate one was also proposed for speed reasons; we applied just the latter one, so we will not describe the first one.

The main idea behind this algorithm is to project each positive example point $x \in X$ outside $X$. To do this, first the set $B$ of the boundary elements of $X$ is calculated,

```
Input: A set of N positive examples (X)
Output: A set of K boundary points (B), and a set of inner points (I)
Init: $B = \emptyset$
1. For each $x$ in $X$ do
2.     Take the $K$ set containing the $k$ closest points to $x$
3.     Compute the minimal and maximal values of the components
       of all the vectors in $K$:
       $(x_{\min})_j = \min_{x_i \in K} ((x_i - x)/\left\|x_i - x\right\|)_j$
       $(x_{\max})_j = \max_{x_i \in K} ((x_i - x)/\left\|x_i - x\right\|)_j$
4.     Form the vector: $x_{center} = x_{\min} + x_{\max}$
5.     For all the vectors in $K$ compute the following values:
       $\cos(\varphi_i) = \frac{(x_i - x)^T \cdot x_{center}}{\|x_i - x\| \cdot \|x_{center}\|}$
6.     If all the $\cos(\varphi_i)$ values are nonnegative,
       then $B = B \cup \{x\}$
```

Table 6.1: The $min/max$-based boundary point search method of the General Counter-example Generator Algorithm

then each positive example is transformed over the closest boundary point, resulting in $N$ negative examples for $N$ positive ones. Next we will describe this algorithm in more detail.

### The Determination of the Boundary Points

First the boundary points of the original example set are calculated; they form a set $B$, where, of course, $B \subseteq X$ (see Table 6.1). It is done by performing the same actions for each $x \in X$. First we find the $k$ closest points to $x$ and arrange them into a set $K$. Then an approximated center vector $x_{center}$ for $x$ is calculated from the extreme values of $K$; that is, for each dimension, the lowest and the highest coordinates are taken, and they are added up after subtracting the appropriate coordinate of $x$ from both of them. Having calculated this center vector $x_{center}$, we calculate the $\cos(\varphi_i)$ values for each $x_i \in K$ via

$$\cos(\varphi_i) = \frac{(x_i - x)^T x_{center}}{\|x_i - x\| \, \|x_{center}\|}. \tag{6.5}$$

If all these values are nonnegative, then $x$ is added to the set of boundary points $B$. As this condition – that the angles between the vectors to the $k$ nearest neighbours and the approximated center vector have to be acute angles – is only a sufficient condition for being a boundary point, the points found by this method are only a subset of the real boundary points. Bánhalmi also suggests an exact solution for this problem by trying to separate $x$ from the elements of $K$ via a Support Vector Machine (SVM) [90], but it is quite apparent that this process is a lot slower than the one described here, which led us to apply the $min/max$ variant.

```
Input: A set of N positive examples (X)
Output: A set of N negative examples (Y)
Init.: $Y = \emptyset$, $B = boundaryPoints(X)$
1. For each $x$ in $X$ do
2.     Find $x_b$, the closest boundary point
       (but with a positive distance) to $x$
3.     Transform $x$ to $x_b$ using the following formula:
       $y = v(1 + T(x, x_b, X)/\left\| v \right\|) + x$, where $v = x_b - x$, T is a
       functional of X, $x$, $x_b$
4.     Check $y$ to see if it is an inner point using the algorithm given in
       Table 6.1
5.     If $y$ is not an inner point
       then $Y = Y \cup \{y\}$
       else $B = B \backslash \{x_b\}$, and with the next closest boundary point
       repeat the procedure.
```

Table 6.2: The method used to generate counter-examples of the General Counter-example Generator Algorithm

### The Projection Over the Closest Boundary Point

Having calculated the $B$ set of boundary points, the next step is to project each element $x \in X$ over the closest boundary point $x_b \in B$, resulting in a new point (hopefully) outside the region of positive examples. The details of this algorithm are given in Table 6.2. Note that it relies on a $T(x, x_b, X)$ function to determine the distance by which the projection in the direction of $x_b - x$ should occur; the proposed function (and the one we applied) is

$$T(x, x_b, X) = \frac{dist}{dist \cdot curv + CosAngle(x, x_b, X)}, \tag{6.6}$$

where

$$CosAngle(x, x_b, X) = \frac{x_{b,center}^T (x - x_b)}{\left\| x_{b,center}^T \right\| \left\| x - x_b \right\|}, \tag{6.7}$$

and $x_{b,center}$ is the center vector for $x_b$ obtained and stored by the boundary point selection method (see Table 6.1). The parameter $dist$ controls the distance of the transformed point and the boundary point, while the $curv$ parameter controls the curvature of the hyper-surface of the counter-examples by applying the transformation on the same boundary point. Fig. 6.2 shows some examples with different $dist$ and $curv$ parameter values.

### Is It Really an Outlier?

There is one last step that should be performed before this standard version of the method terminates. When the transformed, new point is determined, it should be checked to see whether it is indeed an outer point, which is done in the same way as the boundary points were detected (see Table 6.1 once again). If it is not an inner point, then it is added to the set of counter-examples; otherwise the closest boundary point used (i.e. $x_b$) is removed from the set of correct boundaries, and the whole transformation process has to be repeated with the now-closest boundary point $x_b{}' \in B$. Finally this algorithm will generate $N$ counter-examples for a positive dataset of $N$ data samples: one negative example for each positive one.

Note that this algorithm has a rather high time complexity; that is, for $N$ positive examples and $k$ neighbours it is $o(k \cdot N^2 \cdot \log N)$ for the boundary point decision part, and $o(|B| \cdot N \cdot k)$ for the projection. Due to the size of the typical speech recognition databases used we had to make some modifications.

### 6.2.4 Suggested Improvements for the General Counter-Example Generator Algorithm

Having described the three methods from the literature we will test, we are almost ready for the actual experiments. However, the last method (the General Counter-Example Generator Algorithm) has an obvious major weakness: its time requirement for larger databases is very high. As in our case they were indeed large datasets, we had to deal with this issue. The time complexity, even for the $min/max$ boundary point detection variation described here, is of $o(k \cdot N^2 \cdot \log N)$ for $k$ checked neighbours and $N$ points; and in our experiments $N$ was rather big. It meant that executing this algorithm with different $dist$ and $curv$ parameters took rather a long time: even for one particular configuration the counter-example generation ran for about a week on our test machine, which made a thorough testing practically impossible.

We noticed, however, that this algorithm could be divided into two distinct parts: the first one calculates the boundary points and the appropriate center vectors, while the second one is responsible for the actual counter-example generation. Fortunately the time complexity of the first one is responsible for the $o(k \cdot N^2 \cdot \log N)$ part; the $dist$ and $curv$ parameters, however, appear in the second part only. Thus, no matter how many parameter combinations we seek to test, the first – and much slower – part has to be computed just once.

A little later we found that even this second part could be divided up further: when we know the elements of the boundary set $B$, for each element $x \in X$ first we have to find the closest boundary point $x_b \in B$. This procedure also has nothing to do with the varying parameters, thus it can be separated from the actual projection of $x$ over $x_b$, and should also be computed just once (although strictly after determining the boundary elements $B$), then its result can be stored. Thus, for a new $dist$ and $curv$ parameter value pair, we only have to read the index of $x_b$ for each $x$ instead of determining it again as we did in the original version of this algorithm.

After implementing these modifications, we were able to reduce the running time of this algorithm for a parameter pair to about one day. (Of course it involves only the projection of the $x$ points over $x_b$; the precalculation processes took several additional days, but they were performed only once.) It was so because in this part not just the projections are performed, but the resulting point is then examined to decide whether it is an outer point for the positive examples. It is done in the same way as it was in the first part of the algorithm (i.e. taking the $k$ closest points, calculating the $\cos(\varphi_i)$ values, etc.), so it was rather time-consuming; and it cannot be pre-calculated since it is performed on the newly generated counter-example points.

Further investigations led to the observation that this latter part is also not necessarily needed: for larger $dist$ values the projected point lies too far away from the original example set to be an inner point. (At least this was the case in our tests when $dist \geq 1.5$.) Of course the smaller the $dist$ value is, the more likely that some of the projected points may be inner points. But excluding this condition leads to an overwhelming speed-up: this way our counter-example generation procedure was completed in fifteen minutes. Thus we applied this last modification, making it possible to test a large number of parameter pairs.

### 6.2.5 Tests

At this point, having described the problem and the methods we used, we can turn to the testing part. Due to the properties of the anti-phoneme problem, an anti-phoneme modelling method can be evaluated only as a part of the whole speech recognition process. Thus we measured their performance via the standard accuracy and correctness scores achieved by applying them. We will describe the testing methodology of each method applied, and then present the results we obtained.

#### Testing Gaussians

Testing the Gaussian Mixture Models was a rather straightforward task. Our GMM implementation can treat all training elements as the members of only one class, thus no conversion was required to place Gaussian curves over all the phoneme instances. As for the number of Gaussians, we made tests with 10, 15 and 20 components. Since the GMM training procedure is not deterministic due to the initial, clustering part, we performed three tests for each configuration, and then averaged their results.

#### Testing the Incorrect Segment Sampling Algorithm

The testing of this algorithm was also quite straightforward. For each correct phoneme, all six anti-phonemes were generated, and for this set a simple feed-forward ANN was trained with 100 hidden neurons. Next we evaluated this ANN, and the value of the appropriate output neuron served as the approximation of the probability $P(\overline{\alpha}|A_j)$, which was used in the speech recognition process. As the standard way of training a neural net is also not a deterministic one, we performed the training part three times for each set of counter-examples. Then all three were tried out and the performance score was averaged over the three cases.

### Testing the General Counter-example Generation Method

The testing of this method was perhaps the most complicated part as we sought to test several $dist$ and $curv$ pairs to find a successful combination for our actual problem. The testing process for one such parameter pairing is then rather straightforward: we execute the counter-example generator algorithm, and then, similar to the previous case, train an ANN on its output and on the positive examples. As in the previous method, the ANN was a simple feed-forward one with 100 hidden neurons. During an evaluation, the value of the appropriate output neuron served as the estimate for the probability $P(\overline{\alpha}|A_j)$.

To generate the counter-examples we obviously applied the speed-up modifications introduced here. As for the choice of the $dist$ and $curv$ values, first we made some preliminary tests to decide which region of parameter values was worth exploring, with $dist$ values of $1.0$, $2.0$, $3.0$ and $4.0$, and $curv$ values of $0.0$ to $1.0$ with $0.2$ increments. Then a more through testing examined the more promising regions: we generated counter-examples within intervals $1.0 \leq dist \leq 4.0$ and $0.0 \leq curv \leq 0.4$, respectively.

As we used ANNs to separate the positive and the generated negative examples, we again performed the training three times for each set of counter-examples to compensate for the random elements in the ANN training procedure, and took the average of the corresponding values.

## 6.2.6 Results for the Medical Database

We used the Medical Database (see Section 2.5.3) for testing, on which sentence recognition was performed in a segment-based manner. The feature set used was the MFCC-based one used in the SUMMIT system [45]. Under these circumstances, using no anti-phoneme model led to performance scores of $88.17\%$ and $88.53\%$ for accuracy and correctness, respectively. The results of this experiment were published in [35].

### Results for Gaussians

Table 6.3 shows the results we got using GMMs for this problem. There was indeed an improvement in the performance scores, but it cannot be regarded as an overwhelming success. It is also notable that the rate of improvement is rather insensitive to the number of Gaussians used. Overall, an accuracy error rate of $8.88\%$ to $9.02\%$ was attained, meaning a relative error reduction of roughly $25\%$ compared to using no anti-phoneme model at all; the scores for the correctness ratio are $8.20\%$ to $8.52\%$ and $25\%$ to $28\%$, respectively.

### Results for the Incorrect Segment Sampling Algorithm

This method proved to be more effective than the application of GMMs (see Table 6.3); the relative error reduction scores of $32\%$ and $35\%$ seem quite good. Even though this solution is somewhat more difficult to implement, we certainly recommend the use of it in place of GMMs.

| Method | Accuracy | Correctness |
|---|---|---|
| GMM with 10 Gaussians | 91.05% | 91.48% |
| GMM with 15 Gaussians | 90.98% | 91.58% |
| GMM with 20 Gaussians | 91.12% | 91.80% |
| Incorrect Segment Sampling | 91.97% | 92.62% |
| No anti-phoneme modelling | 88.17% | 88.53% |

Table 6.3: Test results for the GMM and the Incorrect Segment Sampling Algorithm on the Medical Database (sentence recognition, segment-based context).

### Results for the General Counter-example Generation Method

This method yielded the best results, although it was the hardest to test. (For the actual accuracy scores see Table 6.4, while for the correctness scores see Table 6.5.) Almost no *dist-curv* pair led to a decrease in the recognition scores, but in most cases the improvement remained at the level of that of the GMMs. At certain points, however, even the method of incorrect segment sampling was significantly outperformed; with $dist = 1.5$ and $curv = 0.0$ we were able to attain scores of $95.58\%$ and $95.82\%$ for accuracy and correctness, respectively. As these values come from averaging the score of three neural networks trained for the same set of examples and counter-examples, this result cannot be just due to chance. These values mean a surprisingly large (over $60\%$) score of relative error reduction compared to having no anti-phoneme model, and $45\%$ to $56\%$ compared to using the incorrect segment sampling method.

Note that this exploration of the domain of the *dist-curv* value pairs would be practically impossible without our proposed speed-up improvements; but these improvements are most likely useful in other contexts as well where counter-example generation is needed and the choice falls to this particular counter-example generator algorithm. Lastly we would like to point out that, similar to the application of operators (see Chapter 5), the actual parameter values ($curv$ and $dist$) which turned out to be the best in our case are of little importance; rather, it is the improvements in the accuracy and correctness scores, which were achieved by setting them properly.

## 6.3 Summary

In this chapter we investigated several strategies for handling the anti-phoneme problem in segment-based speech recognition. As in this task we have to characterize the "phone-ness" of a part of speech when only correct phonemes are known, methods appropriate for one-class classification should be (and were) considered. This chapter covers Thesis II/3, and is covered by publication [35].

Several strategies are available for one-class classification tasks like this: we can model just all the positive examples together as a distribution, or we can somehow generate counter-examples (or negative examples) and we can separate the two classes with a standard classification procedure. The first one, applying GMMs, is the standard

| | | | | *curv* | | |
|---|---|---|---|---|---|---|
| | | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 |
| | 1.0 | **93.91%** | **93.91%** | 90.56% | 91.64% | 91.87% |
| | 1.5 | **95.58%** | 93.31% | 91.40% | 92.47% | 92.59% |
| | 2.0 | 93.19% | 90.80% | 90.92% | 90.20% | **93.43%** |
| *dist* | 2.5 | 90.32% | 90.68% | 89.25% | 91.88% | 91.16% |
| | 3.0 | 90.44% | 90.08% | 91.88% | **94.03%** | 90.80% |
| | 3.5 | 91.52% | 89.13% | 89.13% | 92.35% | 91.87% |
| | 4.0 | **93.67%** | 87.69% | 89.37% | 89.73% | 91.28% |

Table 6.4: Accuracy scores for different *dist* and *curv* parameter values for the General Counter-example Generation Method on the Medical Database (sentence recognition, segment-based context). All scores were averaged over three tests. Notably high values are highlighted in **bold**. Having no anti-phoneme model resulted in a score of $88.17\%$.

way of solving this task in the literature, while in our system the second approach is the default mode.

This second strategy, however, can also be carried out in two ways as counter-examples can be generated in a task-specified manner or in a general one. In our OASIS Speech Laboratory framework the first approach is followed by default, by constructing negative examples from a positive one (i.e. a real phoneme) by changing its starting and/or ending bound. (We called this method the Incorrect Segment Sampling Algorithm.) Our aim in this chapter was to apply a general counter-example generator algorithm in the anti-phoneme subtask in order to improve the overall system performance.

The algorithm chosen was introduced by Bánhalmi [6], and it implemented the idea of projecting each positive example outside the region of positive examples. To do this, first the set of boundary points was determined via some criterion, which was, of course, a subset of the set of positive examples. Then for each positive example the closest boundary point was found, and the point was projected over this boundary. To control this projection two parameters were used: *dist* sets the distance of the resulting point from the original, while *curv* controls the curvature of the hyper-surface of the counter-examples obtained. Finally the resulting point is tested to see whether it is an inner point of the set of positive examples; if it is, the next-closest boundary point has to be determined and the projection has to be repeated, otherwise it becomes a negative example.

It was quite clear that this algorithm was really designed for datasets of a small size (at most $10,000$ elements); in our case, however, the positive examples were all the phonemes of the training database with more than $200,000$ items. As a result, running it with a *dist*-*curv* parameter value pair took quite a lot of time, making a proper parameter setting unfeasible. Luckily we came up with a number of improvements which drastically reduced this time requirement. Several improvements were based on calculating and storing things which had nothing to do with the varying parameters; this way these items did not have to be calculated over and over again when trying

| | | | | $curv$ | | |
|---|---|---|---|---|---|---|
| | | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 |
| $dist$ | 1.0 | **94.26%** | **94.15%** | 91.75% | 92.83% | 93.31% |
| | 1.5 | **95.82%** | 93.67% | 92.35% | 93.67% | 93.91% |
| | 2.0 | 93.55% | 91.16% | 91.28% | 90.80% | **94.02%** |
| | 2.5 | 90.68% | 91.04% | 89.73% | 92.47% | 91.63% |
| | 3.0 | 90.80% | 90.44% | 92.48% | **94.38%** | 91.75% |
| | 3.5 | 91.88% | 89.49% | 89.49% | 92.71% | 92.35% |
| | 4.0 | **94.03%** | 88.05% | 89.73% | 90.56% | 92.11% |

Table 6.5: Correctness scores for different $dist$ and $curv$ parameter values for the General Counter-example Generation method on the Medical Database (sentence recognition, segment-based context). All scores were averaged over three tests. Notably high values are highlighted in **bold**. Having no anti-phoneme model resulted in a score of $88.53\%$.

out various parameter values. In addition, the last step (checking whether the resulting point was an inner point of the set of positive examples) was eliminated for speed reasons and because from our experience we found it was of little practical use. This way this third algorithm could also be applied in the anti-phoneme problem.

As the aim was to improve the actual system performance, we had to run comparative tests for both standard methods and also test the general counter-example generator algorithm. Furthermore, a test without any anti-phoneme model was performed as well to obtain baseline performance. Overall, the standard GMM procedure yielded a $25\%$ decrease in the error rates. The Incorrect Segment Sampling Algorithm performed significantly better by achieving a $32-35\%$ reduction in the error rates. The use of the General Counter-example Generation could lead to a relative error reduction of over $60\%$. Of course it required the adequate setting of its two parameters, but for a method having parameters this is practically inevitable. We should remark here that this parameter-setting procedure would have been practically impossible by simply using the original algorithm due to its time requirements; but after applying our suggested improvements it became a straightforward task.

From these results two main conclusions may be drawn. First, it was worth applying this particular general counter-example generator algorithm in the anti-phoneme task as we could achieve a significant increase in the recognition performance; and second, our improvements made on the basic algorithm were indeed worthwhile, and they should be applied when this algorithm is employed in other tasks as well (within or outside the score of speech recognition).

# Chapter 7

# Conclusions

In the speech recognition problem the constant requirements for improving the speed and levels of accuracy conflict each other. When aiming to speed up the whole recognition process or to raise the recognition accuracy, we always have to pay attention to the other one. In most cases some kind of tradeoff is needed: making a big speed-up while losing a small amount of accuracy, or achieving an improvement in accuracy which requires some further demand on CPU resources. In other cases these problems can be avoided: there are speed-up techniques which do not degrade the recognition performance (or can be configured so that they do not do so), and there are also ideas for achieving higher performance levels which require little or no further CPU time. As this dissertation seeks to make improvements in both areas, it was quite important to follow this second approach and maintain the achievements acquired in one area while making improvements in the other.

For most of the speed-up methods introduced here (namely the multi-pass search in Chapter 3 and the modifications of the multi-stack decoding algorithm in Chapter 4) it was rather easy to do. These were intended just to speed up the system; the only way they could ever affect the recognition accuracy was to decrease it by poor parametrization. To avoid it, these methods were tuned in order to get the maximal speed-up while *retaining the minimal accuracy set*. Had we omitted setting the minimal accuracy level, the achieved speed-up ratios would have had no real meaning anyway due to the trade-off between speed and accuracy. With the segmentation algorithm introduced, however, it was possible to improve both the speed and the accuracy of the speech recognition process at the same time. It seems to be a surprising result at the first glance, but the reason could be that the baseline value itself was the result of this tradeoff: the default segmentation method suggested equidistant bounds, but this distance was set to a certain value. A lower distance (with the appropriate setting of the search method, i.e. larger-sized stacks) leads to a higher recognition accuracy, but also to a (much) higher CPU time demand; compared to this, our proposed segmentation algorithm leads only to speed-up, but to a much larger amount.

The effect on the speed requirements of the accuracy-improving techniques described here is much clearer. Using some kind of operator instead of the traditional product or sum definitely add to the complexity; this addition, however, is so insignificant compared to other stages of the recognition process that it can be easily ignored. As for the anti-phoneme estimation techniques, they of course require more CPU time than using no anti-phoneme model at all, but in practice in a segment-based model this step is rarely omitted. Compared to any of the basic anti-phoneme models used, our proposed algorithm requires actually no further computational time at all during the recognition stage (although it definitely has such demands in the training process).

Finally I would like to mention that a number of techniques found in this dissertation may be of interest for areas outside of speech recognition. First, the logarithmic generator can be used in other fields where finding the optimal triangular norm is required, as well as the technology introduced here to set its parameters. As there are many fields where such fuzzy operators are applied, it could indeed be quite a wide range of areas. And second, when one-class classification is performed, making use of the counter-example generator method outlined here should be considered; and if it is used, then the modifications described in Chapter 6 are clearly worth applying. Thus, these are two methods which were applied in speech recognition in our study, but their potential use goes beyond this field.

# Appendix A

# Algorithms

The pseudocode of the basic search algorithms used are listed next, for which we have to introduce some notations. "$\leftarrow$" 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 $H(t, c, w)$ hypothesis is a triplet of time, cost and a phoneme-sequence. *Extending* a hypothesis $H(t, c, w)$ with a phoneme $v$ and a time $t_i$ results in a hypothesis $H'(t + t_i, c\,', wv)$, where $c\,' = c + c_i$, $c_i$ being the cost of $v$ in the interval $[t, t_i]$. $w(i)$ is the $i$th phoneme of word $w$, $w(i \ldots j)$ is the

## A.1 Stack Decoding Algorithm

This algorithm employs a single (but preferably big) stack, where all the hypotheses are stored. At a given iteration the hypothesis with the lowest cost is popped, it is extended by every possible means, and these newly generated hypotheses are pushed back to the stack. The algorithm ends when a finishing hypothesis is popped.

**Algorithm 1** Stack decoding algorithm

```
$Stack \Leftarrow H_0(t_0, 0, \emptyset)$
while $Stack$ is not empty do
  $H(t_i, c, w) \leftarrow$ top($Stack$)
  if $t_i = t_{\max}$ then
    return $H$
  end if
  for $t_l = t_{i+1} \ldots t_{\max}$ do
    for all $\{v \mid wv \in Pref_{1+\text{length of } w}\}$ do
      $H'(t_l, c', w') \leftarrow$ extend $H$ with $v$ on $[t_i, t_l]$
      $Stack \Leftarrow H'$
    end for
  end for
end while
```

## A.2 Universal A* Algorithm

This is a refinement of the stack decoding algorithm, where for a hypothesis, an estimation is done for the part of the utterance remaining.

**Algorithm 2** Universal A* algorithm

```
$Stack \Leftarrow H_0(t_0, 0, \emptyset)$
while $Stack$ is not empty do
    $H(t_i, c, h, w) \leftarrow$ top($Stack$)
    if $t_i = t_{\max}$ then
        return $H$
    end if
    for $t_l = t_{i+1} \dots t_{\max}$ do
        for all $\{v \mid wv \in Pref_{1+\text{length of } w}\}$ do
            $H'(t_l, c', h', w') \leftarrow$ extend $H$ with $v$ on $[t_i, t_l]$
            $Stack \Leftarrow H'$
        end for
    end for
end while
```

## A.3 Multi-stack Decoding Algorithm

This algorithm employs a separate (but rather small) stack for every possible time indices. Then these stacks are explored advancing in time, and the generated hypotheses are pushed to the stack belonging to their new ending times. The algorithm finishes when all the stacks are explored; then the result will be the finishing hypothesis from the stack of $t_{\max}$ with the lowest cost.

**Algorithm 3** Multi-stack decoding algorithm

```
Stack[$t_0$] $\Leftarrow H_0(t_0, 0, \emptyset)$
for $time = t_0 \dots t_{\max}$ do
    while not empty(Stack[$time$]) do
        $H(t, c, w) \leftarrow$ top(Stack[$time$])
        if $time = t_{\max}$ then
            return $H$
        end if
        for $t_l = time + 1 \dots t_{\max}$ do
            for all $\{v \mid wv \in Pref_{1+\text{length of } w}\}$ do
                $H'(t_l, c', w') \leftarrow$ extend $H$ with $v$ on $[t_i, t_l]$
                Stack[$t_l$] $\Leftarrow H'$
            end for
        end for
    end while
end for
```

## A.4 Viterbi Beam Search

This algorithm is quite similar to the multi-stack decoding one. The only difference is that the stack store not the $n$ best elements, but the best element, plus the ones which have a cost close to the cost of this one.

**Algorithm 4** Viterbi beam search algorithm

```
Stack[$t_0$] $\Leftarrow H_0(t_0, 0, \emptyset)$
for $time = t_0 \dots t_{\max}$ do
  while not empty(Stack[$time$]) do
    $H(t, c, w) \leftarrow$ top(Stack[$time$])
    if $time = t_{\max}$ then
      return $H$
    end if
    for $t_l = time + 1 \dots t_{\max}$ do
      for all $\{v \mid wv \in Pref_{1+\text{length of } w}\}$ do
        $H'(t_l, c', w') \leftarrow$ extend $H$ with $v$ on $[time, t_l]$
        Stack[$t_l$] $\Leftarrow H'$
        Prune Stack[$t_l$] with beam width $T$
      end for
    end for
  end while
end for
```

## A.5 Time-Synchronous Search Algorithm

This algorithm fills a separate table $A$ for every word $w$, where the cell $A(t_i, j)$ holds the hypothesis ending at $t_i$ and having the phoneme sequence $w(1) \dots w(j)$ with the lowest cost. Of course $A(t_{\max}, \text{length of } w)$ will contain the optimal finishing hypothesis for the word $w$, and the chosen word will be the one with the lowest such cost.

## A.6 General Clustering Algorithm

The pseudocode of a general clustering algorithm. "$\leftarrow$" means that a variable is assigned a value. The parameters are: $x_1, x_2, \dots, x_n$ (the initial points to be clustered); a $\mathcal{D}(C_i, C_j)$ distance function between two clusters; and an $L$ value for the stopping criterion. Output: the $C_1, C_2, \dots, C_m$ $(m \leq n)$ clusters.

**Algorithm 5** Time-Synchronous Search Algorithm

```
for all possible word $w$ do
  $A(t_0, 0) \leftarrow H_0(t_0, 0, \emptyset)$
  for $i = t_1 \ldots t_{\max}$ do
    $A(t_i, 1) \leftarrow$ extend $H_0$ with $w(1)$ on $[t_0, t_i]$
  end for
  for $j = 2 \ldots \text{length of } w$ do
    for $i = t_2 \ldots t_{\max}$ do
      $H_{\text{best}} \leftarrow H(t_i, w(1) \ldots w(j), \infty)$
      for $k = t_{j-1} \ldots t_{i-1}$ do
        $H_{\text{act}} \leftarrow A(t_k, j-1)$
        $H_{\text{extended}} \leftarrow$ extend $H_{\text{act}}$ with $w(j)$ on $[t_j, t_k]$
        if $c_{H_{\text{extended}}} < c_{H_{\text{best}}}$ then
          $H_{\text{best}} \leftarrow H_{\text{extended}}$
        end if
      end for
      $A(t_i, j) \leftarrow H_{\text{best}}$
    end for
  end for
  $c(w) = A(t_{\max}, \text{length of } w)$
end for
return $\arg\min_w c(w)$
```

**Algorithm 6** General clustering algorithm

```
for $i = 1, \ldots, n$ do
  $C_i \leftarrow \{x_i\}$
end for
while there is more than one cluster left do
  $(i, j) \leftarrow arg \max \mathcal{D}(C_i, C_j)$ is minimal
  if $\mathcal{D}(C_i, C_j) > L$ then
    break
  end if
  $C_i \leftarrow C_i \cup C_j$
  Remove $C_j$
end while
```

# Appendix B

# Summary in English

Two of the most important aspects of speech recognition are speed and accuracy, as we would like to find the correct transcript of the speech signal uttered, and we would like to find it quickly. Further, there is a trade-off between these two requirements as we can improve the speed of a speech recognizer system if we lower our expectations of accuracy; and if we attempt to raise the accuracy level, it will usually require methods which involve heavy computations. Surprisingly these two aspects have not lost their importance in the development of speech recognition: although ever more powerful machines have appeared, and many smart techniques have became available to improve the recognition accuracy, the expectations have also risen: instead of the early small vocabulary isolated tasks now we have to perform sentence recognition using a large vocabulary. This is why the author decided to concentrate on these two aspects in this dissertation.

Following an introduction to speech recognition, which contains summaries and a general description of the speech recognition problem, possible ways of speeding up the system are presented. Three such methods were discussed, namely a multi-pass search technique, a segmentation algorithm, and a number of improvements for a standard search algorithm. The multi-pass search concept is not a new one: in it, the search process is performed in several steps, each one being more detailed, albeit slower than the previous one. In the variation introduced the earlier passes use a more restricted phoneme set than the latter ones. This way in the earlier passes there are fewer possible words in the vocabulary (as a number of words are fused), and the smaller number of phonemes can be identified faster by the phoneme classifier. In the later passes only the more probable words of the earlier passes remain, so the vocabulary can be reduced there as well. More compact phoneme groups were constructed by clustering the original phoneme set; the distance function used for clustering was calculated from the confusion matrix of the phoneme classifier belonging to the original phoneme set.

The second area investigated where speed-up could be achieved was that of segmentation: in it we restrict the positions in the speech signal where there could be a bound between phonemes (usually done in the segment-based approach). Of course, carelessly omitting possible bounds would reduce the recognition accuracy, while suggesting a lot of unnecessary ones dramatically slows down the search process. This way the speed of speech recognition can clearly be affected, so three algorithms were presented for this, among which one could make the search faster while raising the recognition scores. The remaining speed-up area investigated was the performance of the multi-stack decoding search algorithm, which stores a constant number of hypotheses throughout the whole utterance. By suggesting a number of techniques to dynamically adjust this fixed size, the method could be forced to take the context into account, and it could be speeded up quite significantly.

There are more possibilities for improving the accuracy of speech recognition; we first concentrated on the probability/cost aggregation process. The current statistical approach of speech recognition breaks down the speech signal into smaller and smaller parts; then, after examining them, the appropriate scores are aggregated into higher level values based on the assumption that they are independent. This independence criterion, however, is clearly false, so the operator applied for these aggregations (multiplication) may be replaced if we can find some better-behaving operator. Following this idea, a number of fuzzy functions (mean aggregation operators, triangular norms, and a general triangular norm family) were tested; finally an application-oriented method for modelling the triangular norms was presented, which proved to be the most successful of all.

Lastly the anti-phoneme problem was investigated, where the "phone-ness" of actual speech segments is examined. As there are several positive examples (i.e. correct phonemes), but there is no sure way of having every kind of "non-phoneme" (or "anti-phoneme"), it is an example of one-class classification. For this, besides the two standard methods, a general counter-example generator algorithm was also applied, and a number of techniques were introduced to reduce its time requirements.

As there is a trade-off between speed and accuracy, we had to be very careful while performing the tests. When investigating the speed aspects we took great care not to reduce accuracy; and vice versa, we suggested techniques for improving the accuracy which only had a negligible effect on the running speed.

## B.1 Summary by Chapters

We begin with a short summary of the aims of the dissertation and its key thesis points in Chapter 1. The next chapter (i.e. Chapter 2) is devoted to a description of speech recognition in a top-down manner, breaking up the general problem into ever smaller, more manageable parts. This chapter, of course, does not contain any new results obtained by the author; its task is simply to introduce the notations used, and make the later chapters clearer and more comprehensible. In addition, the methods used for comparing the performance of two speech recognition configurations and a description of the databases used are included.

The remaining chapters describe the work of the author. As every chapter refers to well-known concepts outside of the field of speech recognition, the definitions and description of these things can be also found in this chapter. It led to a standard chapter structure where first the actual subtask of speech recognition is described in more detail, then the corresponding non-speech recognition concepts are clarified. Then we turn to a description of our solution for the given speech recognition subtask, using the ideas mentioned previously. Finally the test environment is described, then the test results are presented and analyzed.

There is one exception to this scheme, namely Chapter 5, which describes the application of different operators in the cost (or probability) aggregation subtask. While this subtask more or less remains the same, quite different operators are applied. There, instead of introducing all of them at the beginning of the chapter, it was done just before their application, each relevant section beginning with a description of a type of operator, outlining its use in the given subtask, and ending with the test results and a discussion.

The order of the chapters themselves more or less follows the chronological order of the author's work. At the beginning he was interested in speeding up the speech recognition process, initially investigating the multi-stack decoding search algorithm. Several ideas were implemented for this purpose, which are covered in Chapter 4. Meanwhile, other speed-up techniques were developed, such as a way of constructing a multi-pass search and new segmentation algorithms, which are described in Chapter 3.

Parallel to the speed-up efforts, the author started investigating the application of various types of aggregation operators in speech recognition. This is described in Chapter 5, in chronological order. First the mean aggregation operators were applied, then the fuzzy triangular norms, which were easier to apply and looked more promising. After utilizing a number of classic triangular norm families (each having one parameter), the author found a way of applying a two-parameter norm and discovered that it was more tunable, and it led to better recognition scores. Afterwards the methodology of modelling a triangular norm even more precisely was developed, which turned out to be the most successful of all norms tested.

In the sixth and final chapter another approach for improving the accuracy of speech recognition is described. It focuses on the anti-phoneme problem, trying to identify the amount of "phone-ness" of the actual speech excerpts. The chapter outlines the basic solution for this task in the literature (i.e. using GMMs), and the basic solution applied in our system (i.e. generating anti-phonemes by choosing incorrect starting and/or ending bounds, then separating the two classes via an ANN). Finally a general counter-example generation method is applied for this task, but to do this, several speed-up modifications had to be introduced. These improvements proved to be very effective for reducing the running time of this algorithm, and the test results show that they did not affect its effectiveness as this method turned out to be the best among the three tested. On the other hand, the modifications introduced were necessary to test a large number of parameter combinations in order to find a proper parameter setting.

## B.2 Key Points of the Thesis

In the following a listing of the most important results of the dissertation is given. Table B.1 summarizes which thesis is described in which publication by the author.

I/1. The author created a multi-pass search technique to speed up the search process in speech recognition. This method was based on the use of multiple, hierarchical phoneme sets, which were created by clustering the elements of the original phoneme set based on the confusion matrix of phoneme classification. Using the hierarchical property of the phoneme groups, the corresponding passes can be readily implemented as a more compact phoneme group fuses several words into one. This way the search process was significantly speeded up. This thesis is explained in detail in Section 3.3.1; it is covered by publications [39; 40; 63].

I/2. The author constructed a method for detecting the interchange of phonemes to more precisely model the bound between them. Based on it, he constructed a novel algorithm to supply a set of possible phoneme boundaries. As shown by the results, this algorithm can indeed lead to a significant speed-up over that of the standard segmentation method, which was our aim. Moreover, a careful configuration of this method also gave a noticeable improvement in the recognition accuracy along with a significant speed-up. This thesis is explained in detail in Section 3.3.2; it is covered by publication [44].

I/3. The author examined the behaviour of the multi-stack decoding algorithm and the Viterbi beam search method. These algorithms both store a number of hypotheses for each possible bound between phonemes. By introducing a number of techniques to more cleverly adjust the number of hypotheses stored, the author was able to speed up the search process along with little or no loss in the recognition accuracy. This thesis is explained in detail in Chapter 4; it is covered by publications [37; 38; 40; 42; 63].

II/1. In the search task of speech recognition the generated hypotheses are ranked by their *cost values*, usually calculated in two steps: first *phoneme-level* costs are aggregated from the *frame-level* ones, then the cost of the hypothesis is calculated from the phoneme-level values. The author applied a number of fuzzy functions at these levels: *mean aggregation operators* and *triangular norms* were tested to improve the recognition accuracy. He also modified the root-power mean operator by introducing a weighting parameter to make earlier observations less important. Furthermore, to set the two parameters of a generalized triangular norm, he turned the problem into a minimization one in a two-dimensional space and applied a global optimization method. The improvements in accuracy indicate that these operators fit the problem better than the default multiplication one. This thesis is explained in detail in Section 5.1; it is covered by publications [36; 38; 40; 41; 62; 63].

II/2. The author proposed an application-oriented method for modelling triangular norms by introducing the logarithmic generator function. Assuming that this function was piecewise linear, he introduced a technique to adequately fit it to the actual application based on the histogram of the occurring probabilities. He applied this method in the speech recognition minimization problem introduced

| | [39] | [44] | [42] | [37] | [38] | [40] | [63] | [41] | [62] | [36] | [43] | [35] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I/1. | ● | | | | | ● | ● | | | | | |
| I/2. | | ● | | | | | | | | | | |
| I/3. | | | ● | ● | ● | ● | ● | | | | | |
| II/1. | | | | | ● | ● | ● | ● | ● | ● | | |
| II/2. | | | | | | | | | | | ● | |
| II/3. | | | | | | | | | | | | ● |

Table B.1: The relation between the theses and the corresponding publications

in Thesis II/1, and showed that it can improve the performance by a greater amount than any other classical norms tested. This thesis is explained in details in Section 5.4; it is covered by publication [43].

II/3. The author applied a general counter-example generator method from the literature in the anti-phoneme problem of segment-based speech recognition. To make it possible to use he also introduced several modifications to this method, which were essential to reduce its time requirement. Doing this, he was able to markedly reduce the error rates of the speech recognition system, even when compared to other standard solutions for this problem. This thesis is explained in detail in Chapter 6; it is covered by publication [35].

# Appendix C

# Summary in Hungarian

A sebesség és a pontosság a beszédfelismerés legfontosabb aspektusai közé tartoznak, mivel a bemondott beszédhang pontos átiratát szeretnénk megkapni, és lehetőség szerint minél hamarabb. Ezen két szempont ráadásul legtöbbször egymás rovására is javítható: gyorsíthatunk a felismerési folyamaton, amennyiben beérjük kisebb pontossággal is, míg a pontosság növelése jellemzően számításigényes módszerek bevezetésével jár. Meglepő módon a beszédfelismerés fejlődése során ezen két szempont mit sem vesztett fontosságából: habár a számítógépek egyre gyorsabbak, és a pontosság növelésére is számos módszert dolgoztak ki, ezzel párhuzamosan az elvárások is megnőttek: a korai kisszótáras, izoláltszavas feladatok helyébe mára nagyszótáras mondatfelismerési problémák léptek. Emiatt a Szerző a beszédfelismerés ezen aspektusaira koncentrált disszertációjában.

A dolgozat egy áttekintéssel és a beszédfelismerési probléma általános ismertetésével kezdődik; ezután először a felgyorsításra bevezetett módszerek kerülnek bemutatásra. Három ilyen eljárásról esik szó: egy többlépéses keresési eljárásról, egy szegmentálóalgoritmusról és egy sor, egy standard keresési módszer felgyorsítására bevezetett technikáról. A többlépéses keresés ötlete nem új: ebben a keresést több lépésben végezzük, melyek mindegyike alaposabb, bár lassabb a korábbiaknál. A konkrét bevezetett módszerben a korábbi lépések szűkebb fonémakészletet használnak, mint az utánuk jövők, ezáltal a korábbi lépések szótára szűkebb (a fonémák egyesítésével egyúttal szavakat is összevonunk), és a kevesebb „fonémát" gyorsabb fonémaosztályozó eljárással lehet azonosítani. A későbbi lépésekben csak az előzőleg valószínűnek talált szavak maradnak a szótárban, így (az eredeti keresőeljáráshoz képest) itt is redukált szótárral dolgozhatunk. A szűkebb fonémacsoportokat az eredeti fonémakészlet klaszterezésével határoztuk meg; a klaszterezés során használt távolságfüggvényt az eredeti fonémakészleten működő fonémaosztályozó eljárás tévesztési mátrixából számítottuk.

A második gyorsításra irányuló terület a szegmentálás feladata volt: ennek során korlátozzuk azokat a pozíciókat, ahol fonémák közti határok lehetségesek. Ezt a feladatot jellemzően szegmensalapú megközelítésben szokás vizsgálni, ott viszont igen sokat számíthat: szükséges határok elhagyása a pontosság csökkenéséhez vezethet, míg fölösleges pozíciók meghagyása nagymértékben lelassíthatja a keresési eljárást. Mivel a megfelelő módszer kiválasztása jelentősen befolyásolhatja a beszédfelismerési eljárás sebességét, három szegmentálóeljárást vezettünk be, melyek közül az egyik nemcsak a sebesség növekedéséhez, hanem ezzel párhuzamosan a felismerési pontosság javulásához is vezetett. Az utolsó gyorsítási terület a multi-stack decoding keresési eljárás vizsgálata volt, mely konstansszámú hipotézist tárol az egész bemondás során. A hipotézisszám rugalmas adaptálására szolgáló technikák bevezetésével a keresési algoritmust sikerült jelentősen felgyorsítani.

A beszédfelismerési pontosság növelésére sokkal több mód kínálkozik; először a valószínűség- vagy költségaggregációs eljárást vizsgáltuk. A beszédfelismerésben jelenleg uralkodó statisztikai megközelítésben a beszédjelet egyre kisebb részekre osztjuk, majd ezek vizsgálata után az előálló valószínűségértékekből magasabbszintű értékeket aggregálunk. Eközben azzal a feltételezéssel élünk, hogy az egyes részek egymástól függetlenek, ez a függetlenségi hipotézis azonban nyilvánvalóan hamis, így az ebből következő műveletet (szorzás) is lecserélhetjük, amennyiben jobban teljesítő operátort találunk. Ezt a gondolatmenetet követve számos fuzzy operátort (középaggregációs operátorokat, háromszögnormákat és egy általános háromszögnorma-családot) teszteltünk; végül bemutattunk egy alkalmazásorientált eljárást a háromszögnormák reprezentálására, mely a legsikeresebbnek bizonyult.

Végül az antifonéma-problémára tértünk át, melyben az előálló beszédszegmensek „fonémaságát" vizsgáljuk. Mivel igen sok pozitív példa (azaz helyes fonéma) áll rendelkezésre, a „nem-fonémák" (vagy „antifonémák") esetében viszont még az sem teljesen egyértelmű, mit kellene figyelembe vennünk, ez a probléma az egyosztályos tanulás körébe tartozik. Erre a feladatra két bevett eljárás kipróbálása mellett egy általános ellenpélda-generáló algoritmust is alkalmaztunk, és az utóbbi időigényének csökkentésére egy sor módosítást is bevezettünk rajta.

Mivel a sebesség és a pontosság egymás rovására is javíthatók, a tesztelés során nagy körültekintéssel kellett eljárnunk. Minden gyorsítási kísérletnél ügyeltünk arra, hogy ne csökkentsük a pontosságot és viszont: olyan pontosságnövelő eljárásokat javasoltunk, melyek a futási időt érdemben nem befolyásolják.

## C.1. A fejezetek áttekintése

A dolgozat céljainak és eredményeinek rövid áttekintésével kezdődik az 1. fejezetben. Ezután a 2. fejezet a beszédfelismerési problémát és annak részfeladatokra bontását írja le. A fejezet természetesen nem tartalmaz semmiféle, a Szerző által kidolgozott eljárást, célja csupán a használt jelölések bevezetése és a későbbi részek érthetőbbé tétele. Emellett két beszédfelismerési konfiguráció összehasonlításának technikái és az adatbázisok leírásai is ott kaptak helyet.

A fennmaradó fejezetek mutatják be a Szerző munkáját. Mivel minden fejezet tartalmaz hivatkozásokat közismert, ám nem beszédfelismerési fogalmakra, ezek leírása szintén a vonatkozó fejezetekbe került. Emiatt egy fejezet jellemzően az adott beszédfelismerési részprobléma részletes leírásával kezdődik, ezután következik a (külső) felhasznált fogalmak bemutatása, majd a bevezetett módszerek ismertetése jön, végül a tesztkörnyezet leírása, a teszteredmények bemutatása és elemzése. Az 5. fejezet, melyben különböző operátorokat alkalmazunk a költség- és valószínűségaggregációs problémában, kivétel ez alól: bár a feladat a fejezet során többé-kevésbé változatlan marad, abban igen eltérő operátorok alkalmazására kerül sor. Emiatt ezeket nem a fejezet elején, hanem közvetlenül alkalmazásuk előtt ismertetjük.

A fejezetek többé-kevésbé időrendi sorrendben követik a Szerző munkáját. Eleinte a beszédfelismerés felgyorsításával foglalkozott, elsősorban a multi-stack decoding keresési algoritmusra koncentrálva; ehhez számos javítási technika került bevezetésre, melyek a 4. fejezetben találhatók. Eközben egyéb gyorsítási módszereket is bevezetett: egy többlépéses keresőeljárás és új szegmentálási módszerek olvashatók a 3. fejezetben.

A gyorsítási kísérletekkel párhuzamosan a Szerző különböző aggregációs operátorok beszédfelismerési alkalmazásával kezdett foglalkozni; ezek leírása az 5. fejezetben található. Először középaggregációs operátorokat használt fel, majd a könnyebben alkalmazható és ígéretesebb fuzzy háromszögnormákat, melyekből klasszikus háromszögnormacsaládokat és egy kétparaméteres, általános háromszögnormát is kipróbált. Ezután bevezetett egy módszert a háromszögnormák eddigieknél rugalmasabb reprezentálására, mely a korábban tesztelt normáknál is sikeresebbnek bizonyult.

Az utolsó, 6. fejezetben is a beszédfelismerési pontosság növeléséről van szó, de az eddigiektől gyökeresen eltérő módon: az antifonéma-problémával foglalkozik, melyben az előforduló beszédszegmensek „fonémaságának" mértékét próbáljuk megbecsülni. Erre a feladatra az irodalomban jellemzően GMM-et használnak, az OASIS rendszerben pedig nem valós kezdő- vagy végpozíciók választásával generálnak antifonémákat. Ezen eljárások mellett a fejezetben a Szerző egy általános ellenpélda-generáló algoritmust alkalmazott a feladatra, melyhez szükség volt egy sor futási időt csökkentő módosítás bevezetésére is. A gyorsításra bevezetett technikák nem befolyásolták az eljárás hatékonyságát, mivel a tesztek során ez bizonyult a leghatékonyabb módszernek; a módosítások viszont szükségesek voltak az alkalmazáshoz, mivel nélkülük nem lehetett volna ilyen nagyszámú tesztet elvégezni, mint amennyi az eljárás paramétereinek megfelelő beállításához szükséges volt.

## C.2. Az eredmények tézisszerű összefoglalása

Az alábbiakban hat tézispontba rendezve összegezzük a Szerző kutatási eredményeit. A kutatásokból származó publikációkat, valamint azok tartalmának az egyes tézispontokhoz való viszonyát a C.1. táblázat tekinti át.

I/1. A szerző létrehozott egy többlépéses keresési eljárást, hogy felgyorsítsa a beszédfelismerési feladat keresési folyamatát. Az eljárás több, hierarchikus fonémacsoportot alkalmazott, melyek az eredeti fonémakészletnek a fonémaosztályozó tévesztési mátrixán alapuló klaszterezésével lettek meghatározva. A fonémacsoportok hierarchikus tulajdonságát kihasználva a megfelelő keresési lépések könnyen implementálhatóak, mivel egy tömörebb fonémacsoportosítás több szót von össze. Ezzel az eljárással sikerült a keresési folyamatot jelentősen felgyorsítani. Ez a tézispont részletesen a 3.3.1. fejezetben van bemutatva, és a [39; 40; 63]

| | [39] | [44] | [42] | [37] | [38] | [40] | [63] | [41] | [62] | [36] | [43] | [35] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I/1. | • | | | | | • | • | | | | | |
| I/2. | | • | | | | | | | | | | |
| I/3. | | | • | • | • | • | • | | | | | |
| II/1. | | | | | • | • | • | • | • | • | | |
| II/2. | | | | | | | | | | | • | |
| II/3. | | | | | | | | | | | | • |

C.1. táblázat. A tézispontok és a Szerző publikációinak viszonya

cikkekben publikálva.

I/2. A szerző megkonstruált egy eljárást a fonémák határainak pontosabb detektálására. Erre alapozva elkészített egy, a fonémák közti lehetséges határokat meghatározó algoritmust. Az eredmények tükrében ez a módszer valóban alkalmazható a beszédfelismerési eljárás felgyorsítására a standard szegmentálási eljáráshoz képest, ezenfelül a módszer pontos paraméterbeállításával a jelentős gyorsítás mellett sikerült a felismerési pontosságon is javítani. Ez a tézispont részletesen a 3.3.2. fejezetben van bemutatva, és a [44] cikkben publikálva.

I/3. A szerző tanulmányozta a multi-stack decoding és a Viterbi beam search kereső eljárásokat, melyek közösek abban, hogy hipotéziseket tárolnak minden lehetséges fonémák közötti határnál. A szerző bevezetett egy sor módszert, melyek a tárolt hipotézisek számát pontonként korlátozták, és így képes volt felgyorsítani a beszédfelismerési probléma keresési folyamatát azonos vagy csak kicsit alacsonyabb felismerési pontosság mellett. Ez a tézispont részletesen a 4. fejezetben van bemutatva, és a [37; 38; 40; 42; 63] cikkekben publikálva.

II/1. A beszédfelismerési probléma keresési feladatában az előálló hipotézisek általában *költségük* alapján vannak rangsorolva, mely érték jellemzően két lépésben számítódik: először *fonémaszintű* költségeket aggregálnak a *keretszintű* értékekből, majd a hipotézisek költségeit határozzák meg a fonémaszintűekből. A szerző fuzzy függvényeket alkalmazott ezen a két szinten: *középoperátorokat* és *fuzzy metszet operátorokat* használt a beszédfelismerés pontosságának növelésére. A korábbi megfigyelések fontosságának csökkentése érdekében bevezette a hatványközép operátor egy kétparaméteres változatát is. Egy felhasznált általánosított fuzzy metszet operátor mindkét paraméterének beállítása érdekében az alkalmazás feladatát átfogalmazta egy kétdimenziós minimalizálási problémává, majd egy globális optimalizáló eljárást használt annak megoldására. A beszédfelismerés pontosságában elért javítások azt jelzik, hogy az alkalmazott operátorok jobban illeszkednek a problémához, mint az alapértelmezett szorzás művelet. A tézispont részletesen a 5.1. fejezetben van bemutatva, és a [36; 38; 40; 41; 62; 63] cikkekben publikálva.

II/2. A szerző kidolgozott egy alkalmazásorientált eljárást a fuzzy metszet operátorok modellezésére a logaritmikus generátorfüggvény bevezetésével. Feltéve, hogy ez a függvény szakaszonként lineáris, bevezetett egy technikát a függvény aktuális problémához illesztésére az előforduló valószínűségek hisztogramjának felhasználásával. Ezt az eljárást a II/1-es tézispontban bevezetett minimalizálási problémában alkalmazta, és megmutatta, hogy nagyobb mértékben képes növelni a beszédfelismerési pontosságot, mint a tesztelt klasszikus normák. A tézispont részletesen a 5.4. fejezetben van bemutatva, és a [43] cikkben publikálva.

II/3. A szerző alkalmazott egy általános ellenpélda-generáló eljárást a szegmensalapú beszédfelismerési feladat antifonéma-problémájában. A felhasználáshoz szükséges volt az eljárás futási idejének radikális csökkentése, aminek elérésére a szerző számos módosítást is bevezetett. Így jelentősen csökkenteni tudta a beszédfelismerési hibát az antifonéma-problémára használt standard eljárásokhoz képest. A tézispont részletesen a 6. fejezetben van bemutatva, és a [35] cikkben publikálva.

# Bibliography

[1] Taheri Asghar, Mohammad Reza Tarihi, Hassan Baghgar Bostan Abad, and Hassan Bababeyk. Fuzzy Hidden Markov Models for speech recognition based on FEM algorithm. In *Proceedings of WEC*, pages 59–61, 2005.

[2] Ikeno Ayako, Bryan Pellom, Dan Cer, Ashley Thornton, Jason M. Brenier, Dan Jurafsky, Wayne Ward, and William Byrne. 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, 2003.

[3] Lalit R. Bahl, P.S. Gopalakrishnan, and Robert L. Mercer. Search issues in large vocabulary speech recognition. In *Proceedings of the 1993 IEEE Workshop on Automatic Speech Recognition*, Snowbird, UT, 1993.

[4] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:179–190, 1983.

[5] Todd M. Bailey and Ulrike Hahn. Phoneme similarity and confusability. *Journal of Memory and Language*, 52(3):339–362, April 2005.

[6] András Bánhalmi, András Kocsor, and Róbert Busa-Fekete. Counter-example generation-based one-class classification. In *Proceedings of ECML*, pages 543–550, 2007.

[7] András Bánhalmi, Dénes Paczolay, László Tóth, and András Kocsor. Development of a Hungarian medical dictation system. *Informatica*, 31(2):241–246, 2007.

[8] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[9] Francesco Beritelli, Luca Borrometi, and Antonino Cuce. Phoneme fuzzy characterization in speech recognition systems. In *Proceedings of SPIE*, volume 3165, pages 305–310, 1997.

[10] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[11] András Bánhalmi, Dénes Paczolay, Lászó Tóth, and András Kocsor. Development of a hungarian medical dictation system. *Informatica*, 31:241–246, 2007.

[12] Henve Bourland, Hynek Hermansky, and Nelson Morgan. Towards increasing speech recognition error rates. *Speech Communication*, 18:205–231, 1996.

[13] Henve Bourland, Yochai Konig, and Nelson Morgan. REMAP: recursive estimation and maximization of a posteriori probabilities – application to transition-based connectionist speech recognition. *ICSI Technical Report TR-94-064*, 1994.

[14] Anthony Brew, Marco Grimaldi, and Pádraig Cunningham. An evaluation of one-class classification techniques for speaker verification. *Artificial Intelligence Review*, 27(4):295–307, 2007.

[15] Ciprian Chelba. *Exploiting Syntactic Structure for Natural Language Modeling*. PhD thesis, Johns Hopkins University, Maryland, 2000.

[16] Michael J. Cloud and Bryon C. Drachman. *Inequalities*. Springer-Verlag, 1998.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*, chapter 13: Red-Black Trees, pages 273–301. MIT Press and McGraw-Hill, 2001.

[18] Frederick J. Damerau. *Markov Models and Linguistic Theory: An Experimental Study of a Model for English*. The Hague: Mouton, 1971.

[19] Pierre A. Devijver and Josef Kittler. *Pattern Recognition, a Statistical Approach*. Prentice Hall, 1982.

[20] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[21] József Dombi. A general class of fuzzy operators, the de morgan class of fuzzy operators and fuzziness measures induced by fuzzy operators. *Fuzzy Sets and Systems*, 8:149–163, 1982.

[22] József Dombi. Towards a general class of operators for fuzzy systems. *IEEE Transaction on Fuzzy Systems*, 16(2):477–484, 2008.

[23] Jasha Droppo, Li Deng, and Alex Acero. Uncertainty decoding with splice for noise robust speech recognition. In *Proceedings of ICASSP*, pages 307–310, Orlando, Florida, 2002.

[24] Didier Dubois and Henri Prade. *Fundamentals of Fuzzy Sets*. Kluwer Academic Publisher, 2000.

[25] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley & Sons, New York, 1973.

[26] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons Inc., 2001.

[27] Gunnar Evermann, Ho Yin Chan, Mark J.F. Gales, Thomas Hain, Xunying Liu, David Mrva, Lan Wang, and Philip C. Woodland. 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.*, pages 249–252, Montreal, Canada, 2004.

[28] János Fodor and Marc Roubens. *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer Academic Publisher, 1994.

[29] Jean-Luc Gauvain, Lori Lamel, Gauvain Adda, and Martine Adda-Decker. The limsi continuous speech dictation system: Evaluation on the arpa wall street journal task. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '94)*, pages 557–560, 1994.

[30] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models*. Cambridge University Press, 2002.

[31] James R. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech and Language*, 17(2):137–152, 2003.

[32] James R. Glass, Jane Chang, and Michael McCandless. A probabilistic framework for feature-based speech recognition. In *Proceedings of the 1996 International Conference on Spoken Language Processing*, pages 2277–2280, Philadelphia, PA, 1996.

[33] Joshua Goodman. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 11–25, 1997.

[34] Géza Gordos and György Takács. *Digital Speech Processing (in Hungarian)*. Műszaki Könyvkiadó, 1983.

[35] Gábor Gosztolya, András Bánhalmi, and László Tóth. Using one-class classification techniques in the anti-phoneme problem. In *Proceedings of the 2009 Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, volume LNCS 5524, pages 433–440, Porto, Portugal, 2009.

[36] Gábor Gosztolya, József Dombi, and András Kocsor. Applying the Generalized Dombi Operator family to the speech recognition task. *Journal of Computing and Information Technology*, 17(3):285–293, 2009.

[37] Gábor Gosztolya and András Kocsor. Improving the multi-stack decoding algorithm in a segment-based speech recognizer. In *Proceedings of the 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE)*, volume LNCS 2718, pages 744–749, Loughborough, England, UK, 2003.

[38] Gábor Gosztolya and András Kocsor. Aggregation operators and hypothesis space reductions in speech recognition. In *Proceedings of the 2004 Conference on Text, Speech and Dialogue (TSD)*, volume LNCS 3206, pages 315–322, Brno, Czech Republic, 2004.

[39] Gábor Gosztolya and András Kocsor. A hierarchical evaluation methodology in speech recognition. *Acta Cybernetica*, 17(2):213–224, 2005.

[40] Gábor Gosztolya and András Kocsor. Speeding up dynamic search methods in speech recognition. In *Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE)*, volume LNCS 3533, pages 98–100, Bari, Italy, 2005.

[41] Gábor Gosztolya and András Kocsor. Using triangular norms in a segment-based automatic speech recognition system. *International Journal of Information Technology and Intelligent Computing (IT & IC) (IEEE)*, 1(3):487–498, 2006.

[42] Gábor Gosztolya, András Kocsor, László Tóth, and László Felföldi. Various robust search methods in a Hungarian speech recognition system. *Acta Cybernetica*, 16 (2):229–240, 2003.

[43] Gábor Gosztolya and László L. Stachó. Aiming for best fit t-norms in speech recognition. In *Proceedings of the 2008 International Symposium on Intelligent Systems and Informatics (SISY) (IEEE)*, pages 1–5, Subotica, Serbia, Sept 2008.

[44] Gábor Gosztolya and László Tóth. Detection of phoneme boundaries using spiking neurons. In *Proceedings of the 2008 International Conference on Artificial Intelligence and Soft Computing (ICAISC)*, volume LNCS 5097, pages 782–793, Zakopane, Poland, 2008.

[45] Andrew K. Halberstadt and James R. Glass. Heterogeneous measurements and multiple classifiers for speech recognition. In *Proceedings of the 1998 International Conference on Spoken Language Processing (ICSLP)*, pages 995–998, Sydney, Australia, 1998.

[46] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. The MIT Press, 2001.

[47] Godfrey Harold Hardy, John Edensor Littlewood, and György Pólya. *Inequalities*. Cambridge University Press, 1968.

[48] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to "A formal basis for the heuristic determination of minimal cost paths". In *SIGART Newsletter, No. 37*, pages 28–29, 1972.

[49] Wolfgang Herbordt, Toshiharu Horiuchi, Masakiyo Fujimoto, Takatoshi Jitsuhiro, and Satoshi Nakamura. Noise-robust hands-free speech recognition and communication on pdas using microphone array technology. In *Proceedings of the IEEE*

*Workshop on Automatic Speech Recognition and Understanding*, pages 307–310, San Juan, Puerto Rico, 2005.

[50] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing*. Prentice Hall, 2001.

[51] Waltraud Huyer and Arnold Neumaier. Snobfit – stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software*, 35(2):1–25, 2008.

[52] Frederick Jelinek. A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13:675–685, 1969.

[53] Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1997.

[54] Sándor Jenei and Endre Pap. Smoothly generated archimedean approximation of continuous triangular norms. *Fuzzy Sets and Systems (Special Issue "Triangular norms")*, 104:19–25, 1999.

[55] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*, chapter 6: N-grams, pages 191–234. Prentice Hall, New Jersey, 2000.

[56] Stephan Kanthak, Hermann Ney, Michael Riley, and Mehryar Mohri. A comparison of two LVR search optimization techniques. In *Proceedings of the 2002 International Conference on Spoken Language Processing (ICSLP)*, pages 1309–1312, Denver, CO, 2002.

[57] Nikola Kasabov, Robert Kozma, Richard Kilgour, Mark Laws, Michael J. Watts, Andrew R. Gray, and John G. Taylor. *Speech Data Analysis and Recognition Using Fuzzy Neural Networks and Self-Organised Maps*, pages 241–263. Physica Verlag, 1999.

[58] Zoltán Kató. Segmentation of color images via reversible jump MCMC sampling. *Image Vision Comput.*, 26(3):361–371, 2008.

[59] Brian E.D. Kingsbury. *Perceptually-inspired Signal Processing Strategies for Robust Speech Recognition in Reverberant Environments*. PhD thesis, University of California at Berkeley, 1998.

[60] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Kluwer Academic Publisher, 2000.

[61] András Kocsor, András Bánhalmi, and Dénes Paczolay. Technological background of a speech recognizer system working on scintigraphy records (in Hungarian). *Acta Agraria Kaposvariensis*, 10(1):113–128, 2006.

[62] András Kocsor and Gábor Gosztolya. Application of full reinforcement aggregation operators in speech recognition. In *Proceedings of the 2006 Conference of Recent Advances in Soft Computing (RASC)*, Canterbury, UK, 2006.

[63] András Kocsor and Gábor Gosztolya. The use of speed-up techniques for a speech recognizer system. *The International Journal of Speech Technology*, 9 (3-4):95–107, 2006.

[64] András Kocsor and László Tóth. Kernel-based feature extraction with a speech technology application. *IEEE Transactions on Signal Processing*, 52(8):2250–2263, 2004.

[65] András Kocsor, László Tóth, and Jr. András Kuba. An overview of the OASIS speech recognition project. In *Proceedings of the 1999 International Conference on Applied Informatics*, Eger-Noszvaj, Hungary, 1999.

[66] Ron Kohavi and Foster Provost. Glossary of terms. *Editorial for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process*, 30 (2/3), February/March 1998.

[67] Moshe Koppel and Jonathan Schler. Authorship verification as a one-class classification problem. In *Proceedings of the 21th International Conference on Machine Learning (ICML)*, pages 489–495, Banff, Alberta, Canada, 2004.

[68] Heungkyu Lee and Hanseok Ko. Competing models-based text-prompted speaker independent verification algorithm. *Speech Communication*, 48:28–44, 2006.

[69] Kai-Fu Lee. *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer, Dordrecht, The Netherlands, 1989.

[70] Steven C. Lee and James R. Glass. Real-time probabilistic segmentation for segment-based speech recognition. In *Proceedings of the 1998 International Conference on Spoken Language Processing (ICSLP)*, pages 347–358, Sydney, Australia, 1998.

[71] Hank Liao and Mark J. F. Gales. Issues with uncertainty decoding for noise robust automatic speech recognition. *Speech Communication*, 50(4):265–277, 2008.

[72] Yang Liu, Holger Jones, Sheila Vaidya, Michael Perrone, Borivoj Tydlitát, and Ashwini K. Nanda. Speech recognition systems on the cell broadband engine processor. *IBM Journal of Research and Development*, 51(5):583–591, 2007.

[73] David M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, 1994.

[74] Mathworks. Matlab, 1984-2008. http://www.mathworks.com.

[75] Péter Mihajlik, Tibor Fegyó T, B Németh, Z Tüske, and V Trón. Towards automatic transcription of large spoken archives in agglutinating languages: Hungarian ASR for the MALACH Project. In *Proceedings of the 2007 International Conference on Text, Speech and Dialogue (TSD)*, pages 342–350, Pilsen, Czech Republic, 2007.

[76] Nelson Morgan and Henve Bourland. An introduction to hybrid HMM/connectionist continuous speech recognition. *Signal Processing Magazine*, May 1995:1025–1028, 1995.

[77] Géza Németh, Gábor Olaszy, Mátyás Bartalis, Géza Kiss, Csaba Zainkó, and Péter Mihajlik. Speech based drug information system for aged and visually impaired persons. In *Proceedings of the International Conference on Speech and Technology (Interspeech)*, pages 2533–2536, Antwerp, the Netherlands, 2007.

[78] Hermann Ney and Stefan Ortmanss. Progress in dynamic programming search for LVCSR. *Proceedings of the IEEE*, 88(8):1224–1240, 2000.

[79] Long Nguyen, Richard Schwartz, Francis Kubala, and Paul Placeway. Search algorithms for software-only real-time recognition with very large vocabularies. In *Proceedings of the 1993 Workshop on Human Language Technology (HLT)*, pages 91–95, Morristown, NJ, USA, 1993. Association for Computational Linguistics. ISBN 1-55860-324-7.

[80] Atsunori Ogawa, Yoshiaki Noda, and Shoichi Matsunaga. A second-pass search algorithm for multi-pass speech recognition strategy. *Joho Shori Gakkai Kenkyu Hokoku*, 2000(15):51–56, 2000.

[81] Mari Ostendorf. Moving beyond the 'beads-on-a-string' model of speech. In *In Proc. IEEE ASRU Workshop*, pages 79–84, 1999.

[82] Mari Ostendorf, Vassilis Digalakis, and Owen A. Kimball. From HMMs to segment models: A unified view of stochastic modeling for speech recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing, Volume 4*, pages 360–378, 1996.

[83] Janne Pylkkönen and Mikko Kurimo. uration modeling techniques for continuous speech recognition. In *Proceedings of the 8th International Conference on Spoken Language Processing (Interspeech 2004)*, pages 385–388, Jeju Island, Korea, 2004.

[84] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[85] Gailius Raškinis and Danutė Raškinienė. Building medium-vocabulary isolated-word lithuanian HMM speech recognition system. *Informatica*, 14(1):75–84, 2003.

[86] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, pages 97–104. Prentice Hall, 2003.

[87] Tara N. Sainath. Acoustic landmark detection and segmentation using the McAulay-Quatieri sinusoidal model. Master's thesis, MIT, 2005.

[88] Ruhi Sarikaya, Yuqing Gao, and Hakan Erdogan. Turn-based language modeling for spoken dialog systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '02)*, pages 781–784, Orlando, Florida, USA, 2002.

[89] Jeffrey C. Schlimmer. Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the International Conference on Machine Learning '93*, pages 284–290, 1993.

[90] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[91] Christian Schrumpf, Martha Larson, and Stefan Eickeler. Syllable-based language models in speech recognition for english spoken document retrieval. In *Proceedings of AVIVDiLib*, pages 196–205, Cortona, Italy, 2005.

[92] Richard Schwartz, Long Nguyen, and John Makhoul. *Multiple-pass Search Strategies*, chapter 18, pages 429–456. Kluwer Academic Publisher, Philadelphia, PA, 1996.

[93] Berthold Schweizer and Abe Sklar. Associative functions and statistical triangle inequalities. *Publ. Math. Debrecen*, 8:169–186, 1961.

[94] Györgyi Sejtes and András Kocsor. The database specification of SpeechMaster. *Alkalmazott Nyelvtudomány*, IV(1):81–89, 2004.

[95] Máté Szarvas, Tibor Fegyó, Péter Mihajlik, and Péter Tatai. Automatic recognition of Hungarian: Theory and practice. *International Journal of Speech Technology*, 3:237–252, 2000.

[96] David M.J. Tax. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, 2001.

[97] David M.J. Tax and Robert P.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[98] László Tóth and András Kocsor. Explicit duration modelling in HMM/ANN hybrids. In *Proceedings of the 2005 Conference on Text, Speech and Dialogue (TSD)*, pages 310–317, Karlovy Vary, Czech Republic, 2005.

[99] László Tóth, András Kocsor, and János Csirik. On Naive Bayes in speech recognition. *International Journal of Applied Mathematics and Computer Science*, 15 (2):287–294, 2005.

[100] László Tóth, András Kocsor, and Gábor Gosztolya. Telephone speech recognition via the combination of knowledge sources in a segmental speech model. *Acta Cybernetica*, 16(4):643–657, 2004.

[101] László Tóth, András Kocsor, and Kornél Kovács. A discriminative segmental speech model and its application to Hungarian number recognition. In *Proceedings of the 2000 Conference on Text, Speech and Dialogue (TSD)*, pages 307–313, Brno, Czech Republic, 2000.

[102] Dat Tran and Michael Wagner. Fuzzy Expectation-Maximisation algorithm for speech and speaker recognition. In *Proceedings of NAFIPS*, pages 421–425, 1999.

[103] Dat Tran, Michael Wagner, and T. VanLe. A proposed decision rule based on fuzzy c-means clustering for speaker recognition. In *Proceedings of the 1998 International Conference on Spoken Language Processing (ICSLP)P*, volume 2, pages 755–758, 1998.

[104] Petre Tzvetkov, Xifeng Yan, and Jiawei Han. Tsp: Mining top-k closed sequential patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03*, pages 347–354. IEEE Press, 2003.

[105] Klára Vicsi and Attila Víg. Language independent automatic segmentation technique using SAMPA labelling of phonemes. In *Proceedings of the First International Conference on Language Resources Education*, pages 1317–1323, Granada, Spain, 1998.

[106] Klára Vicsi, András Kocsor, Csaba Teleki, and László Tóth. Hungarian speech database for computer-using environments in offices (in Hungarian). In *Proceedings of MSZNY 2004*, pages 315–318, Szeged, Hungary, 2004.

[107] Klára Vicsi, László Tóth, András Kocsor, and János Csirik. MTBA – a Hungarian telephone speech database (in Hungarian). *Híradástechnika*, LVII(8), 2002.

[108] Balázs Visy, Klára Vicsi, and László Kosik. Generic dialogue system for phone information systems. In *High Speed Networking International Workshop*, pages 21–24, Balatonfüred, Hungary, 1997.

[109] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. *Sun Microsystems Technical Report TR-2004-139*, 2004.

[110] Ye-Yi Wang and Alex Waibel. Decoding algorithm in statistical machine translation. In *Proceedings of the 8th Conference on European Chapter of the Association for Computational Linguistics*, pages 366–372, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[111] Steve Young. *The HMM Toolkit (HTK) (software and manual).* http://htk.eng.cam.ac.uk/, 1995.

[112] Steve Young. Statistical modelling in continuous speech recognition. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, Seattle, 2001.