Iterative Segmentation in Keyword Spotting: relation to fractional programming

Marius C. Silaghi
Florida Institute of Technology

April 15, 2005

Technical Report CS-2005-13

Abstract

This paper addresses the problem of detecting keywords in unconstrained speech. The proposed algorithms search for the speech segment maximizing the average observation probability[1] along the most likely path in the hypothesized keyword model. As known, this approach (sometimes referred to as sliding model method) requires a relaxation of the begin/endpoints of the Viterbi matching, as well as a time normalization of the resulting score. This makes solutions complex (i.e., $\frac{LN^2}{2}$ basic operations for keyword HMM models with L states and utterances with N frames).

We present here two alternative (quite simple and efficient) solutions to this problem. a) First we provide a method that finds the optimal segmentation according to the criteria of maximizing the average observation probability. It uses Dynamic Programming as a step, but does not require scoring for all possible begin/endpoints. While the worst case remains $O(LN^2)$, this technique converged in at most $3(L+2)N$ basic operations in each experiment for two very different applications. b) The second proposed algorithm does not provide a segmentation but can be used for the decision problem of whether the utterance should be classified as containing the keyword or not (provided a predefined threshold on the acceptable average observation probability). This allows the algorithm to be even faster, with fix cost of $(L+2)N$.

[1]The accumulated log posterior (or likelihood) divided by the length of the segment.

1 Introduction

This paper addresses the problem of *keyword spotting (KWS)* in unconstrained speech. Typical applications are search in audio databases and recognition of spoken passwords, but due to their general framework the proposed algorithms also apply to computer vision, natural language processing, and bioinformatics [4]. Detecting the beginning and the end of the keyword segment in the utterance is called *segmentation*. This is typically done by explicit modeling of non-keyword segments using a filler Hidden Markov Model (HMM) or an ergodic HMM composed of context dependent or independent phone models without lexical constraints [18]. The utterance is then modeled with HMMs where the keyword is preceded and followed by non-keyword segments. The use of such non-keyword models, trained from speech databases, offers no guarantee that the hypothesized keyword segment maximizes any rigorous measure of the matching with the keyword model. Some slower approaches to KWS, without explicit modeling of non-keyword segments, have a stronger theoretical foundation. Namely, they maximize a chosen measure of the match between the keyword segment and a keyword model.

Although several algorithms tackling KWS without models of non-keyword segments have already been proposed in the past, (e.g., by using Dynamic Time Warping [7] or Viterbi matching [19, 5, 10, 2] allowing relaxation of the begin and endpoint constraints), these are known to require the use of an "appropriate" normalization of the matching scores since segments of different lengths have then to be compared. At any possible ending time e, the match score of the best warp and start time b of the reference has to be computed [7] (for all possible start times b associated with unpruned paths). Moreover, in [19], and in the same spirit than what is presented here, for all possible ending times e, the average observation likelihood along the most likely state sequence is used as scoring criterion. Sometimes these techniques are referred to as "sliding model methods" and need $\frac{LN^2}{2}$ updates to elements of a $L{\times}N$ Dynamic Programming (DP) phone trellis with L HMM states per keyword and utterances with N time frames. Previous attempts to construe faster alternatives based on straightforward DP [10] use cost functions that do not respect DP's optimality principle [1] and are therefore sub-optimal from the point of view of the pursued measures.

Here we will use a similar rigorous scoring technique for keyword spotting like the one used by algorithms without filler models. We use an explicit filler model, re-estimated at each utterance in a way proven to yield optimal segmentation (for the chosen measure). Our matching score is based on the average observation probability (AOP)[2]. The first algorithm proposed here, besides computing the AOP score for the occurrence of the hypothesized keyword into an utterance, also produces the corresponding segmentation – the decision of where the keyword starts and ends. Compared to previously devised "sliding model" methods (such as [7, 19]), the algorithms proposed here re-estimate a filler model, using Viterbi decoding which does not require scoring for all begin/endpoints, and which can be proved to (quickly) converge to the "optimal solution" (from the point of view of the chosen scoring function, i.e., AOP). Each re-estimation step uses straightforward Viterbi alignments (similar to regular filler-based KWS).

2 Measures and Notations

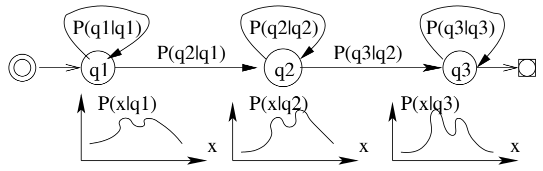

Figure 1: A simple progressive left-to-right HMM.

A continuous Hidden Markov Model (HMM) [9, 15], see Figure 1, is defined by a set of states $\mathcal{Q} = \{q_1, \ldots, q_L\}$ with a unique starting state, a space of possible outputs (for speech recognition, typically the space of Mel-frequency cepstrum coefficients feature vectors), a set of transition probabilities $P(q_i|q_j), \forall i, j \in [1..L]$, and a set of probability density functions defining the likelihood[3] of each possible output for each state.

Let $X = \{x_1, \ldots, x_N\}$ denote the sequence of acoustic vectors in which we want to detect a keyword, and let M be the HMM model of a keyword and consisting of L states $\mathcal{Q} = \{q_1, \ldots, q_L\}$ where q_1 is the starting state and q_L

[2]likelihoods can sometimes be used instead of posteriors

[3]or posterior

the final one. Assume that M is matched to a subsequence $X_b^e = \{x_b, \ldots, x_e\}$ ($1 \leq b \leq e \leq N$) of X, and that we have an implicit (not modeled) *garbage/filler state* q_G preceding and following M. Thus we implicitly introduce the grammatical constraint that we recognize one keyword occurrence, preceded and followed by non-keyword segments.

2.1 Average Observation Posterior

The average observation posterior (AOp) of a model M given a subsequence X_b^e is defined as the average of the log posterior probability along the optimal path, i.e.:

$$\begin{aligned} AOp(M, X, b, e) &= \frac{1}{e-b+1} \min_{Q \in M} -\log P(Q|X_b^e) \\ &= \frac{1}{e-b+1} \min_{Q \in M} \{-\log P(q^e|x_e) \\ &- \sum_{n=b}^{e-1} [\log P(q^n|x_n) + \log P(q^{n+1}|q^n)]\} \end{aligned} \tag{1}$$

where $Q = \{q^b, q^{b+1}, ..., q^e\}$ represents any of the possible paths of length $(e-b+1)$ in M, and q^n the HMM state visited at time n along Q, with $q^n \in \mathcal{Q}$. The local posteriors $P(q_k|x_n)$ can be estimated as output values of a multilayer perceptron (MLP) as used in hybrid HMM/Neural Network systems [6].

For a specific sub-sequence X_b^e, expression (1) can easily be estimated by dynamic programming since the subsequence and the associated normalizing factor $(e-b+1)$ are given and constant. However, in the case of keyword spotting, this expression should be estimated for all possible begin/endpoint pairs $\{b, e\}$ (as well as for all possible word models), and we define the matching score of X and M as:

$$AOp(M, X) = AOp(M, X, b_p^*, e_p^*) \tag{2}$$

where the optimal begin/endpoints $\{b_p^*, e_p^*\}$, are the ones yielding the lowest average local log posterior

$$\langle b_p^*, e_p^* \rangle = \underset{\{b,e\}}{\operatorname{argmin}} \ AOp(M, X, b, e) \tag{3}$$

Given the time normalization and the relaxation of begin/endpoints, straightforward DP is no longer optimal and has to be adapted, usually involving more CPU. A new (and simple) solution to this problem is proposed in this paper.

3 Background

Let us summarize the techniques that compute the same output as the algorithms proposed here or/and that are used as intermediary steps in our techniques.

3.1 Filler-based KWS

Although various solutions have been proposed towards the direct optimization of (2) as, e.g., in [7, 19], most of the KWS approaches today prefer to preserve the simplicity of Viterbi DP by modeling the complete input X [13]. This is done by explicitly [14] or implicitly [5] modeling non-keyword segments, using so called filler or garbage models as additional reference models. We assume that non-keyword segments are modeled by extraneous garbage models/states q_G (and grammatical constraints ruling the possible keyword/non-keyword sequences).

In this paper, we will consider only the case of detecting one keyword per utterance at a time. In this case, the keyword spotting problem amounts at matching the whole sequence X of length N onto an extended HMM model $\overline{M}$ consisting of the states $\{q_0, q_1, \ldots, q_L, q_{L+1}\}$, in which q_0 and q_{L+1} are tied (i.e., share emission probabilities) to the garbage model state q_G. A path of length N in $\overline{M}$ (whose segment contained in M is $Q = \{q^b, ..., q^e\}$) is denoted $\overline{Q} = \{\overbrace{q_0, ...q_0}^{b-1}, q^b, q^{b+1}, ..., q^e, \overbrace{q_{L+1}, ...q_{L+1}}^{N-e}\}$ with $(b-1)$ garbage states q_0 preceding q^b and $(N-e)$ states q_{L+1} following q^e, and respectively emitting the vector sequences X_1^{b-1} and X_{e+1}^N associated with the non-keyword segments.

Given some estimation ε_t of $-\log P(q_G|x_n)$, the optimal path $\overline{Q_t}$ (and, consequently segmentation b_t and e_t) using the posteriors criteria is then given by:

$$\begin{aligned}
\overline{Q_t} &= \underset{\forall \overline{Q} \in \overline{M}}{\text{argmin}} -\log P(\overline{Q}|X) \\
&= \underset{\forall \overline{Q} \in \overline{M}}{\text{argmin}} \{-\log P(Q|X_b^e) \\
&\quad +(b-1)\log P(q_0|q_0) + (N-e)\log P(q_{L+1}|q_{L+1}) \\
&\quad -\sum_{n=1}^{b-1} \log P(q_G|x_n) - \sum_{n=e+1}^{N} \log P(q_G|x_n)\}
\end{aligned} \tag{4}$$

which can be solved by straightforward DP, all paths having the same length (see Algorithm 1). The main problem of filler-based keyword spotting approaches is then to find ways to best estimate $P(q_G|x_n)$ in order to minimize the error introduced by the approximations. In [5] this value is set to the average of the K best local scores (for some value K). In other approaches it is generated by training explicit filler HMMs. These approaches will usually not lead to the "optimal" solutions given by (2).

Viterbi decoding retrieves the most probable path through a HMM M that generates a certain observation X. Its first step uses a DP algorithm computing the probability of the best path through M given X (i.e., Equation 4). This DP in the first step of Viterbi is isolated in Algorithm 1 (ViterbiScoring), which returns the score of the best path. It uses an array V storing in each $V[i,j]$ the score of the best path of length i in M that starts at the start state and ends in state q_j. It is typically prefered to use local posteriors for $Prob[i,j]$ in Algorithm 1. Nevertheless, some systems use local likelihoods [9, 6].

```
function ViterbiScoring($M$, $X$, $Prob$)
    foreach $k$ from 2 to $L$, $V[1,k] = \infty$;
    $V[1,1] = Prob[1,1]$;
    foreach $i$ from 2 to $N$ do
        $u = i\%2$; // index $i$ is used $mod$ 2 to save space;
        foreach $j$ from 1 to $L$ do
            $V[u,j] = Prob[i,j] +$
                $\min_{k \in [1..L]} (V[1-u,k] - \log P(q_j|q_k))$;
    return $V[N\%2, L]$;
```

Algorithm 1: ViterbiScoring is part of the first DP step of the Viterbi algorithm. $Prob[i,j]$ can be $-\log P(x_i|q_j)$ instead of $-\log P(q_j|x_i)$.

3.2 Sliding Model Method

The sliding model method [19], see Algorithm 2, returns the Average Observation Probability (posterior or likelihood) of the best match of the keyword model with a segment of the observation. This is done by computing the score for each segment. It reuses the partial results between segments starting at the same frame. If the basic operation is the computation at Line 2.1, then the cost is $LN(N-1)/2$. $\{b^*, e^*\}$ denote the optimal segmentation.

4 Segmentation by Filler Re-estimation

In the following we define a method referred to as *Segmentation by Filler Re-estimation (SFR)* with good/fast convergence properties, estimating a value for $-\log P(q_G|x_n)$ (or $-\log P(x_n|q_G)$ when using likelihoods) such that straightforward DP for computing (4) yields exactly the same segmentation (and recognition results) than (3). While the same result is achieved through the sliding model method the algorithm proposed below is more efficient.

SFR is based on the same HMM structure as the filler based approaches (4). However, rather than looking for explicit (and empirical) estimates of $P(q_G|x_n)$ we aim at mathematically estimating its value (which will be different

```
**function SlidingModel**(M, X, Prob)
    $score = \infty$;
    **foreach** $b$ from $1$ to $N{-}1$ **do**
        **foreach** $k$ from $2$ to $L$, $V[b\%2, k] = \infty$;
        $V[b\%2, 1] = Prob[b, 1]$;
        **foreach** $i$ from $b{+}1$ to $N$ **do**
            $u = i\%2$;
            **foreach** $j$ from $1$ to $L$ **do**
2.1             $V[u, j] = Prob[i, j] + \min_{k \in [1..L]} (V[1{-}u, k] - \log P(q_j|q_k))$;
            $score = \min(score, \frac{V[u,L]}{(i-b+1)})$;
            **if** $score$ changed **then** $b* \leftarrow b$, $e* \leftarrow i$;
    return $(score, b*, e*)$;
```

Algorithm 2: Sliding Model Method.

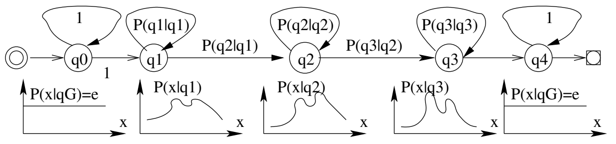

Figure 2: Keyword with Filler Model for SFR

```
**function SegmFillReest**(M, X, Prob, $\varepsilon$)
    **foreach** $k$ from $1$ to $L{+}1$ **do** $V[0, k] = \infty$;
    $V[0, 0] = 0$;
    **while** true **do**
3.1     **foreach** $i$ from $1$ to $N$ **do**
            $u = i\%2$;
            $Prob[i, 0] = Prob[i, L + 1] = \varepsilon$;
            **foreach** $j$ from $0$ to $L{+}1$ **do**
                $v = \operatorname{argmin}_{k \in [0..L+1]} (V[1{-}u, k] - \log P(q_j|q_k))$;
3.2             $b[u, j] = (v == 0) ? i : b[1{-}u, v]$;
                $V[u, j] = Prob[i, j] + (V[1{-}u, v] - \log P(q_j|q_v))$;
3.3         $e[u] = (v \leq L) ? i : e[1{-}u]$;
3.4     $\varepsilon = \frac{V[u,L+1] - \varepsilon N}{e[u] - b[u,L+1]} + \varepsilon$;
        **if** $\varepsilon$ unchanged **then break**;
    return $(\varepsilon, b[u, L{+}1], e[u])$;
```

Algorithm 3: Segmentation by Filler Re-estimation. The ε parameter may take any value. Theoretically the algorithm may be faster if ε is closer to the result. Experimentation shown that it made little difference in practice.

and adapted to each utterance) such that solving (4) is equivalent to solving (3). Here q_G is a special emitting state whose emission probability density function is not normalized, but is constant (see Figure 2). Also, the transition probabilities from/to garbage states are not normalized (equaling 1). As such, continuing to use the probability notations with (4) would be an abuse of language and we rewrite it using ε instead of $-\log P(q_G|x_n)$, ε having only an algebraic significance. The estimation of ε at step t is denoted ε_t.

$$\overline{Q_t} = \underset{\forall \overline{Q} \in \overline{M}}{\operatorname{argmin}} \{-\log P(Q|X_b^e) + (b-1+N-e)\varepsilon_t\} \tag{5}$$

Thus, we perform a converging re-estimation of ε, such that the segmentation resulting of (5) is the same than what would be obtained from (3).

SFR can be summarized as follows (see Algorithm 3):

1. Start from an initial value[4] ε_0, (e.g., with ε_0 equal to 0 or to a cheap estimation of the score of a "match").
2. Find $\overline{Q_t}$ (and b_t, e_t) according to (5) (Lines 3.1 to 3.3).
3. Update the estimated value of ε to the average of the local posteriors along the keyword segment of the path Q_t, (Line 3.4) i.e.,:
$$\varepsilon_{t+1} = AOp(M, X, b_t, e_t) = \frac{-1}{(e_t - b_t + 1)} \log P(Q_t|X_{b_t}^{e_t})$$
4. If $\varepsilon_{t+1} \neq \varepsilon_t$, then increment t and return to Step 2.

Lemma 1 (AOp(M,X) is a fix point) *If $\varepsilon_t = AOp(M, X)$ given by (2), then solving (4) yields $\langle Q^*, b^*, e^*\rangle$ given by (3).*

We note from the definition of AOp that $\forall\ b, e$,

$$AOp(M, X, b, e) \geq AOp(M, X, b^*, e^*). \tag{6}$$

Lemma 2 (decrease if $\varepsilon_t >$ AOp(M,X)) *If $\varepsilon_t > AOp(M, X)$, then the path computed with (5) is $\overline{Q_t}$ with $AOp(M, X, b_t, e_t) < \varepsilon_t$.*

Theorem 1 *The series $\{\langle Q_t, b_t, e_t\rangle\}_t$, defined with the recursion $\varepsilon_{t+1} = AOp(M, X, b_t, e_t)$ where Q_t, b_t, e_t are computed from ε_t with (5), converges to the unique fix point $\langle Q^*, b^*, e^*\rangle$.*

Lemma 3 (shrinking path) *If $\varepsilon_t > \varepsilon_{t+1}$, then either $e_{t+1} - b_{t+1} < e_t - b_t$ or we have reached convergence.*

The proof of these propositions appears in Appendix: each SFR cycle (from the second one) decreases the estimation of ε, and the final path yields the same solution than (3).

Corollary 1.1 *SFR converges in maximum N cycles.*

Proof. From the previous lemma and from Lemma 2 we notice that at each step before convergence (starting with the 2^{nd}), the length of Q_t decreases. The number of steps is therefore roughly upper bounded by N. □

Since SFR is proven to terminate in at most N cycles, its proven worst cost is $N^2(L+2)$ basic operations, where the basic operation in SFR adds Line 3.2 to the basic operation of the Algorithm 2. While SFR's theorethical worst case is more than twice worse than the fix (best case) cost of Algorithm 2, SFR always converged in 3 cycles in the experiments described further, leading to an expected cost of $3N(L+2)$, which is much better (N's value is between 176 and 1097 in our database).

4.1 Relation with 0-1 Fractional Programming

Another way of proving the convergence of SFR (which yields weaker bounds for termination) consists in proving the equivalence of SFR with a slightly less efficient version where ε is subtracted in each cycle from each element of $Prob$ (see Algorithm 4). We now prove that this new version is equivalent to an instance of Dinkelbach's procedure [8, 16, 12]'s for a *zero-one fractional programming problem* with real coefficients.

[4]In Appendix, it is actually proven that SFR will always converge to the same solution (in more or less cycles, with the worst case upper bound of N loops) independently of this initialization.

```
function SegmFillSubst($M$, $X$, $Prob$, $\varepsilon$)
    foreach 2 from 1 to $L$ do $V[0,k] = \infty$;
    $V[0,1] = 0$;
    while true do
        foreach $i$ from 1 to $N$ do
            $u = i\%2$;
            foreach $j$ from 1 to $L$ do
                $v = \operatorname{argmin}_{k\in[1..L]} (V[1{-}u,k] - \log P(q_j|q_k))$;
                $b[u,j] = (v == 0)?i : b[1{-}u,v]$;
4.1             $V[u,j] = Prob[i,j] - \varepsilon +$
                        $(V[1{-}u,v] - \log P(q_j|q_v))$;
            $e[u] = (v \leq L)?i : e[1{-}u]$;
        $\varepsilon = \frac{V[u,L+1]}{e[u]-b[u,L+1]} + \varepsilon$;
        if $\varepsilon$ unchanged then break;
    return $(\varepsilon, b[u,L+1], e[u])$;
```

Algorithm 4: Segmentation by Filler Substraction.

```
procedure FP do
    1. Pick a random $\vec{x}$.
    2. Compute $\lambda = \frac{P(\vec{x})}{Q(\vec{x})}$.
    3. Estimate $\vec{x} = \operatorname{argmax}_{\vec{x}} P(\vec{x}) - \lambda Q(\vec{x})$.
       If $0 \neq \max_{\vec{x}} P(\vec{x}) - \lambda Q(\vec{x})$ then go to Step 2, else return $\langle \vec{x}, \lambda \rangle$.
```

Algorithm 5: Dinkelbach's procedure for fractional programming

Zero-one fractional programming (0-1 FP) [12] solves the optimization over n variables

$$\operatorname*{argmax}_{\vec{x}\in W} \frac{P(\vec{x})}{Q(\vec{x})}.$$

Here $\vec{x} = \{x_1, ..., x_n\}$, $\vec{x} \in W$ where $W \subseteq \{0,1\}^n \setminus \{0\}^n$, $P(\vec{x}) = \sum_{P\in A} a_P \prod_{i\in P} x_i + \sum_{i=1}^n a_i x_i$, $Q(\vec{x}) = \sum_{P\in B} b_P \prod_{i\in P} x_i + \sum_{i=1}^n b_i x_i$. $A \subseteq \mathcal{P}([1..n])$ and $B \subseteq \mathcal{P}([1..n])$ for $\mathcal{P}([1..n]) = \{P|P \subseteq [1..n], |P| \geq 2\}$, $a_P \geq 0$, $b_P \leq 0$, $a_i \geq 0$, $b_i > 0$.

Dinkelbach's procedure for fractional programming is based on the observation that $\operatorname*{argmax}_{\vec{x}} \frac{P(\vec{x})}{Q(\vec{x})} = \{\vec{x}|\exists\lambda; \max_{\vec{x}} P(\vec{x}) - \lambda Q(\vec{x}) = 0\}$, and is shown in Algorithm 5. Dinkelbach's procedure is known to converge in $n+1$ iterations.

Let us formalize the keyword spotting problem based on AOP using 0-1 fractional programming. First we define $n = NL$ variables in $\{0,1\}$, namely $x_{0,0}, ..., x_{L,N}$, where $x_{k,t} = 1$ if the phone represented by HMM state q_k is the one actually generating the time frame t. Each variable corresponds to an element of the X-M lattice and states whether that element is visited by the best match.

A is the set of all possible transitions in the lattice denoted by a triplet. Each $\{i,j,k\} \in A$ corresponds to a possible transition from state q_j at time i to state q_k at time $i{+}1$. $a_{\{i,j,k\}} = -\log P(q_k|q_j)$. $a_{i,j} = -\log P(q_j|X_i^i)$ is the minus log probability of being in state q_j at time i given the utterance X. $B = \emptyset$ and $b_{i,j} = 1$. W is the set of possible assignments of $\vec{x}$ that set to 1 exactly the elements on a paths through the HMM M that generate some subsequence of X. The obtained 0-1 FP problem is:

$$\underset{\vec{x}\in W}{\operatorname{argmin}} \left\{ \frac{\sum_{i=1,j=1}^{i=N,j=L} -\log P(q_j|X_i^i)x_{j,i} + \sum_{i=1,j=1,k=1}^{i=N-1,j=L,k=L} -\log P(q_k|q_j)x_{j,i}x_{k,i+1}}{\sum_{i=1,j=1}^{i=N,j=L} x_{j,i}} \right\}$$

It can be easily observed that this problem defines the AOP criteria in Equation 2. When Dinkelbach's procedure for solving this FP is run using the Algorithm 1 for the estimation at Step 3, one arrives exactly at Algorithm 4. According to [12], this algorithm converges in maximum $n + 1 = LN + 1$ iterations. Our result in Corollary 1.1 is stronger, proving convergence in at most N iterations (this is due to the properties of this problem).

5 Decision by Filler Re-estimation

A typical usage of keyword spotting is to detect the presence of some keyword in an utterance (without caring for the segmentation). If the score of the matching is above some threshold T, then a positive detection is declared.

Lemma 4 *When the estimation of ε is initialized with T, if it increases at the first iteration then the match will be rejected, otherwise it will be accepted.*

Proof. From (6) it follows that whenever ε_0 has a smaller value than the fix point, the first re-estimation will transform it in a value bigger than the fix point, i.e., $\varepsilon_1 > \varepsilon_0$. If ε_0 has a larger value than the fix point, the first re-estimation will yield a smaller value, $\varepsilon_1 < \varepsilon_0$ (Lemma 2). Therefore the direction of change in the first re-estimation of ε tells if the initialization value was smaller or bigger than the fix point.

If we only want to know how T compares to the fix point (which is the score of the match), it follows that we can learn this in one cycle by setting ε_0 to T. $\square$

```
function DecisionFillReest($M, X, Prob, T$)
    foreach $k$ from 1 to $L$+1 do $V[0,k] = \infty$;
    $V[0,0] = 0$;
    foreach $i$ from 1 to $N$ do
        $u = i\%2$;
        $Prob[i,0] = Prob[i,L+1] = T$;
        foreach $j$ from 0 to $L$+1 do
            $V[u,j] = Prob[i,j] +$
                $\min_{k\in[0..L+1]} (V[1-u,k] - \log P(q_j|q_k))$;
    return $V[u,L] > TN$;
```

Algorithm 6: Decision by Filler Re-estimation.

The obtained algorithm is called *Decision by Filler Re-estimation (DFR)* (see Algorithm 6) and roughly consists of a cycle of SFR. The basic operation (defined as for the sliding model method) is performed only $N(L+2)$ times.

6 Fractional Programming for Double Normalization

A double averaging of the probability of a match with first the number of frames per phone and then the number of phones in the keyword was known to yield good accuracy [3, 17]. Assuming 1 state per phone, this is:

$$S_{M,X}(Q,b,e) \stackrel{\text{def}}{=} \frac{-1}{L}\sum_{j=1}^{L}\left(\frac{\sum_{n=b_j}^{e_j}\log P(q_j^n|x_n)}{e_j - b_j + 1}\right) \tag{7}$$

where L represents the number of phones in M and q_j^n the hypothesized phone q_j for input frame x_n. The quality of a match increases with the decrease of this score.

As with the AOP criteria, this problem can be solved exactly using 0-1 fractional programming. The FP program for this case is:

$$\operatorname*{argmin}_{\vec{x}\in W}\left\{\frac{\sum_{j=1}^{j=L}\left(\prod_{t=1,t\neq j}^{t=L}\sum_{i=1}^{i=N}x_{j,i}\right)\left(\sum_{i=1}^{i=N}-\log P(q_j|X_i^i)x_{j,i}+\sum_{i=1}^{i=N-1}-\log P(q_j|q_j)x_{j,i}x_{j,i+1}\right)}{\prod_{j=1}^{j=L}\sum_{i=1}^{i=N}x_{j,i}}\right\}$$

Dinkelbach's FP procedure cannot use dynamic programming (or Viterbi) for the optimization in Step 3. Instead, we can use the graph minimum-cut algorithm proposed in [12].

Lemma 5 *Dinkelbach's procedure converges in maximum $N-L$ iterations for the double normalization problem.*

Proof. It is shown in [12] that the number of variables set to 1 decreases in each iteration. Since in our problem this number corresponds to the length of the chosen path, and since the maximum length is N and the minimum length is L, it follows that the maximum number of iterations is $N-L$.

7 Experimental results

Our algorithms and the sliding model method were compared on the BREF database [11], a continuous, read speech microphone database. 242 utterances (with a $2,300$ word lexicon), were used for testing. These keywords were represented by simple hybrid HMM/ANN models based on context-independent phones.

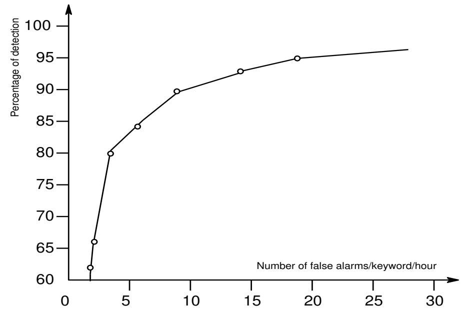

Figure 3: *ROC of the SFR-based keyword detection based on (2) as a function of number of false alarms/keyword/hour, as obtained 100 keywords selected at random.*

On this database, the average speedup of SFR (13.867.986 DP updates/keyword) vs. sliding model (1.241.899.326 updates/keyword) is 89.5 times. The resulting ROC (Receiver Operating Characteristics) curve, using SFR to estimate (2), is presented in Figure 3. SFR gives the same results as the sliding model method (confirming the soundness of the implementation). For computing the segmentation, 3 iterations were needed in each experiment. While it may be believed that the fast convergence was due to lucky choices of databases and keywords, the same fast convergence was observed for all keywords in the database and for also for a very different application to natural language parsing of another version of our algorithm [4].

DFR is SFR which stops after the first (out of 3) iterations. DFR is obviously 3 times faster than SFR and yields the same result. For building a ROC curve it is preferable to use SFR since DFR would have to be run again for each computed point of the curve. However, for production systems (working at a predefined point) DFR is preferable.

We do not provide here the comparison between recognition with AOP scoring versus other scorings, since these comparisons have been thoroughly studied by other authors [18, 4]. In a summary, some other scores work better than AOP with certain keywords and databases, and vice-versa. AOP's advantage is that it is not a black-box but a rigorous measure and its behavior improves understanding of keyword spotting.

8 Conclusions

We have thus proposed new methods for keyword spotting (KWS), working with both local likelihoods and local posterior probabilities, and optimizing rigourously defined matching scores (the average observation probability) between keyword models and keyword segments. The fastest previous algorithm with this property is Sliding Model requiring exactly $N^2L/2$ basic operations, a prohibitive cost.

A first proposed algorithm, *Segmentation by Filler Re-estimation (SFR)*, computes the same output like Sliding Model but much faster: $3N(L+2)$ measured in two very different experimented applications, KWS and NLP, even if its theorethical worst case cost is $O(N^2L)$. A second proposed algorithm, called *Decision by Filler Re-estimation (DFR)* returns the same matching decision like SFR and sliding model methods, but does not compute the segmentation (which is not required in many applications). However its cost is fix, $N(L+2)$ of the same basic operations like in sliding model method. Proof is given in Appendix. Typical applications are search in audio databases and recognition of spoken passwords, but due to its use of the general HMM framework, the proposed algorithms have been shown to have applications in computer vision, natural language processing, and bioinformatics [4].

A Appendix

Lemma 1 (AOp(M,X) is a fix point) *If $\varepsilon_t = AOp(M,X)$ given by (2), then solving (4) yields $\langle Q^*, b^*, e^* \rangle$ given by (3).*

Proof. Let us note $AOp(M,X)$ with w. We see from definitions that $-\log P(\overline{Q^*}|X) = N * w$. A suboptimal path $\overline{Q_b^e}$ would have yielded $-\log P(\overline{Q_b^e}|X) = N * w + (e-b+1) * (w' - w) > N * w$ since $w' > w$ from the definition of AOp. Here we have noted by w' the value of the average log probability in the suboptimal path Q_b^e, i.e.:

$$w' = AOp(M, X, b, e) = \frac{1}{e-b+1} - \log P(Q_b^e | X_b^e) > w$$

Since DP yields the optimum, it chooses $\overline{Q^*}$. □

Lemma 2 (decrease if $\varepsilon_t >$ AOp(M,X)) *If $\varepsilon_t > AOp(M,X)$, then the path computed with (5) is $\overline{Q_t}$ with $AOp(M,X,b_t,e_t) < \varepsilon_t$.*

Proof. Let us denote $AOp(M,X,b_t,e_t)$ with w' and $AOp(M,X)$ with w.

$$-\log P(\overline{Q_t}|X) = (N - 1 - e_t + b_t) * \varepsilon_t + (e_t - b_t + 1) * w'$$

(i.e., contribution of non-keyword segments '+' contribution of keyword segment). Since the path $\overline{Q^*}$ was not preferred:

$$-\log P(\overline{Q_t}|X) \leq -\log P(\overline{Q^*}|X)$$

This can be written as: $(N-1-e_t+b_t) * \varepsilon_t + (e_t - b_t + 1) * w' \leq (N - 1 - e^* + b^*) * \varepsilon_t + (e^* - b^* + 1) * w$.

After reordering we obtain: $(e_t - b_t + 1) * w' \leq (N - 1 - e^* + b^*) * \varepsilon_t + (e^* - b^* + 1) * w - (N - 1 - e_t + b_t) * \varepsilon_t$, and by regrouping the right term:

$$(e_t - b_t + 1) * w' \leq (e^* - b^* + 1) * w + ((e_t - b_t) - (e^* - b^*)) * \varepsilon_t \tag{8}$$

Since $e^* - b^* + 1 > 0$ and from the hypothesis that $w < \varepsilon_t$:

$$(e^* - b^* + 1) * w + ((e_t - b_t) - (e^* - b^*)) * \varepsilon < (e_t - b_t + 1) * \varepsilon_t \tag{9}$$

From the inequalities (8) and (9):

$$(e_t - b_t + 1) * w' < (e_t - b_t + 1) * \varepsilon_t$$

$e_t - b_t + 1 > 0 \Rightarrow w' < \varepsilon_t$.

If Q^* was preferred, then again $w' = w < \varepsilon_t$. □

Theorem 1 *The series $\{\langle Q_t, b_t, e_t\rangle\}_t$, defined with the recursion $\varepsilon_{t+1} = AOp(M, X, b_t, e_t)$ where Q_t, b_t, e_t are computed from ε_t with (5), converges to the unique fix point $\langle Q^*, b^*, e^*\rangle$.*

Proof. From inequality (6), after the 2^{nd} step we have $\varepsilon \geq AOp(M, X)$. From Lemma 2, each further cycle of the algorithm decreases ε.

From Lemma 1 and Lemma 2, results that the convergence appears at $AOp(M, X, b_t, e_t) = AOp(M, X)$. Since the number of possible paths is finite, it converges. □

Lemma 3 (shrinking path) *If $\varepsilon_t > \varepsilon_{t+1}$, then either $e_{t+1} - b_{t+1} < e_t - b_t$ or we have reached convergence.*

Proof. Let us denote $w_t = AOp(M, X, b_t, e_t)$ and $w_{t+1} = AOp(M, X, b_{t+1}, e_{t+1})$. Since $\overline{Q_t}$ was preferred to $\overline{Q_{t+1}}$ for $\varepsilon = \varepsilon_t$, it means that $\varepsilon_t(b_t - e_t + N - 1) + w_t(e_t - b_t + 1) \leq \varepsilon_t(b_{t+1} - e_{t+1} + N - 1) + w_{t+1}(e_{t+1} - b_{t+1} + 1)$.
$\varepsilon_t((e_{t+1} - b_{t+1}) - (e_t - b_t)) + (e_t - b_t + 1)w_t - (e_{t+1} - b_{t+1} + 1)w_{t+1} \leq 0$ (12)

If for $\varepsilon = \varepsilon_{t+1}$ ($\varepsilon_{t+1} < \varepsilon_t$), we would obtain $\overline{Q_{\varepsilon_{t+1}}}$ with $e_{t+1} - b_{t+1} \geq e_t - b_t$, $(e_{t+1} - b_{t+1}) - (e_t - b_t) \geq 0$, then by adding $(\varepsilon_{t+1} - \varepsilon_t)((e_{t+1} - b_{t+1}) - (e_t - b_t)) \leq 0$ to inequality (12) we obtain:
$\varepsilon_{t+1}((e_{t+1} - b_{t+1}) - (e_t - b_t)) + (e_t - b_t + 1)w_t - (e_{t+1} - b_{t+1} + 1)w_{t+1} \leq 0$
showing that $\overline{Q_{\varepsilon_t}}$ will still be preferred to $\overline{Q_{\varepsilon_{t+1}}}$. If $\overline{Q_{\varepsilon_t}} \neq \overline{Q_{\varepsilon_{t+1}}}$ then this is a contradiction and we can infer that $e_{t+1} - b_{t+1} < e_t - b_t$. Otherwise we have $\overline{Q_{\varepsilon_t}} = \overline{Q_{\varepsilon_{t+1}}}$ and we reach convergence with $e_{t+1} - b_{t+1} = e_t - b_t$. □

References

[1] R. Bellman. On the theory of dynamic programming. *Proc. of the National Academy of Sciences*, 38:716–719, 1952.

[2] Y. Benayed, D. Fohr, J.P. Haton, and G. Chollet. Confidence measures for keyword spotting using suport vector machines. In *ICASSP*, 2003.

[3] G. Bernardis and H. Bourlard. Improving posterior based confidence measures in hybrid hmm/ann speech recognition systems. In *Proceedings to ICSLP'98 (Sydney, Australia)*, 1998.

[4] Authors Blinded, citation.

[5] J.-M. Boite, H. Bourlard, B. d'Hoore, and M. Haesen. A new approach towards keyword spotting. In *EUROSPEECH*, pages 1273–1276, 1993.

[6] H. Bourlard and N. Morgan. *Connectionist Speech Recognition - A Hybrid Approach.* Kluwer, 1994.

[7] J.S. Bridle. An efficient elastic-template method for detecting given words in running speech. In *Brit. Acoust. Soc. Meeting*, pages 1–4, 1973.

[8] J.R. Isbell and H. Marlow. Attrition games. *Naval Res. Logist. Quart*, 3:71–93, 1956.

[9] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1999.

[10] J. Junkawitsch, L. Neubauer, H. Höge, and G. Ruske. A new keyword spotting algorithm with pre-calculated optimal thresholds. In *ICSLP*, pages 2067–2070, Philadelphia, PA, 1996.

[11] L.-F. Lamel, J.-L. Gauvain, and E. Eskénazi. Bref, a large vocabulary spoken corpus for french. In *Proceedings of EuroSpeech'91*, 1991.

[12] Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12:141–159, 1982.

[13] J.R. Rohlicek. Word spotting. In R.P. Ramachandran and R. Mammone, editors, *Modern Methods of Speech Processing*, pages 123–157. Kluwer, 1995.

[14] R.C. Rose and D.B. Paul. A hidden markov model based keyword recognition system. In *ICASSP'90*, pages 129–132, 1990.

[15] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[16] W.Dinkelbach. On nonlinear fractional programming. *Management Science*, 18(7):492–498, Mar. 1967.

[17] G. Williams and S. Renals. Confidence measures for hybrid hmm/ann speech recognition. In *Proceedings of Eurospeech*, pages 1955–1958, 1997.

[18] J. Wilpon, L. Rabiner, C. Lee, and E. Goldman. Automatic recognition of keywords in unconstrained speech using hidden markov models. *IEEE Trans. on ASSP*, 38(11):1870–1878, November 1990.

[19] J.G. Wilpon, L.R. Rabiner, Lee C.-H., and E.R. Goldman. Application of hidden markov models of keywords in unconstrained speech. In *Proc. of ICASSP'89*, pages 254–257, 1989.