THEORY OF STRUCTURED COGITION
IN SPEECH RECOGNITION

Václav Hanžl
Czech Technical University, Faculty of Electrical Engineering
Technická 2, 166 27 Prague 6, Czech Republic
hanzl@noel.feld.cvut.cz
http://noel.feld.cvut.cz/hypo

ABSTRACT

We propose general representation of knowledge and hypotheses used in speech recognition systems. General abstract entity is described and further specialised to represent various types of information – time intervals, speech signal, sequences of Markov model states, pronunciation, written text etc.

These types of data represent tiers of an overall knowledge about the utterance and during the recognition process not all the tiers are fully known. Furthermore, mutual alignment of individual tiers is not fully known. Operations with these multi-tier partially known structures have common definition on the abstract level and individual datatypes are created by further restrictions of the basic abstraction. Common properties are however strong enough to enable design of algorithms optimizing the whole recognition system.

1. INTRODUCTION

In earlier work [1] we described general hypothesis driven engin for speech recognition systems. This article presents an overview of the theoretical approach to datastructures used in the system.

Many types of information present in speech recognition systems can be regarded as individual tiers of the overall information about the utterance. This is quite obvious for the written text tier which contains information about the result of recognition and for the phoneme sequence tier which contains sequence of units used somehow during the recognition. Speech sound itself forms another tier of this structure. In the task of recognition, the sound tier is known at the beginning of recognition, other tiers are unknown. We are trying to make the text tier known and for this purpose we are also making known other intermediate tiers and alignment of the tiers.

Fundamental property of a tier is the possibility to know it only partially – to know for example only beginning of a sentence while the rest is yet unknown.

Several tiers aligned at the beginning and at the end (at so called sequence points) form a hypothesis. We will use graphical representation to depict this. For example the following hypothesis says that phoneme “e” is present in the speech signal starting at time 5.123 seconds and ending at time 5.327 seconds:

<table>
<thead>
<tr>
<th>5.123</th>
<th>5.327</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td></td>
</tr>
</tbody>
</table>

and this one means that the word “zpěv” pronounced as a phoneme sequence “spej” is present from time 4.462 to time 5.633 seconds:

<table>
<thead>
<tr>
<th>4.462</th>
<th>5.633</th>
</tr>
</thead>
<tbody>
<tr>
<td>zpev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spej</td>
</tr>
</tbody>
</table>

Some tiers might be unknown; we will use question marks to express this. Thus a hypothesis with known pronunciation but unknown written form might be:

<table>
<thead>
<tr>
<th>?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spej</td>
</tr>
</tbody>
</table>

Uncomplete hypothesis is a question, more complete one is an answer. The above hypothesis may be answered by several ways:

<table>
<thead>
<tr>
<th>zpjev</th>
<th>zpjev</th>
<th>zpev</th>
<th>zpev</th>
</tr>
</thead>
<tbody>
<tr>
<td>spej</td>
<td>spej</td>
<td>spej</td>
<td>spej</td>
</tr>
</tbody>
</table>

Hypotheses can be combined together using binary operations called inferences and coalescences. Operands of inferences remain in the database of hypotheses (called hypobase), operands of coalescence are replaced by the result. Any of these operations might happen once both the operands are ready, however the system tries to use good hypotheses first. This rather free definition of operation order makes very effective parallelization possible, on the other hand the recognition process is not an algorithm in the strict sense - repeated computation with the same input data might lead to different results. In practice the speed gain achieved by this setup is very high and the result uncertainty negligible – result of recognition might not be perfect anyway.

Basic operations (all either inferences or coalescences) are serial concatenation, overlay and inverse decomposition operations. Example overlay operation might combine recognised phoneme string with pronunciation rule to get hypothesis containing result.
of recognition bound to particular times:

\[
\begin{pmatrix}
3.12 & 3.74 \\
sp & sp \\
3.12 & 3.74 \\
zpev & zpev \\
sp & sp 
\end{pmatrix}
\] →

\[
\begin{pmatrix}
3.12 & 3.74 \\
sp & sp \\
3.12 & 3.74 \\
zpev & zpev \\
sp & sp 
\end{pmatrix}
\]

Example concatenation operations connect e.g. words to form parts of sentences and whole sentences.

2. CONCATENABLE TIERS

Most notable abstract property of a tier is the possibility to concatenate corresponding tiers of two hypotheses to form new "longer" hypotheses. For some types of tiers this operation is quite obvious and it is just concatenation of strings. We can however define the "concatenation" for other things as well – for probabilities we define the concatenation operation as a product and get exactly the right behaviour needed in Viterbi decoding. For context dependent strings the concatenation operation might provide no results when there is not a context match. For time intervals the ending time of first hypothesis must match the beginning time of the second one.

Hypotheses composed of tiers serve as information storage units, questions, answers, input data and output data.

3. VITERBI DECODING

The traditional Hidden Markov Model recognition can be emulated using our approach. Let us suppose that HMM has probabilities of transition from state \( i \) to state \( j \) given by \( a_{i,j} \) and probability that an observation vector \( o \) is emitted when entering state \( j \) is given by function \( b_j(o) \). Our example Markov model of phoneme "e" might look like this:

![Diagram of a Markov model with states and transitions](image)

We will use these tiers: time, sequence of HMM states and probability.

We will use hypotheses of four types: type A – represents transition between states, type B – represents entering a state and emitting an observation vector, type PATHB – represents whole path from the beginning of the model till entering a state, and type PATHA – represents whole path from the beginning till a transition (but not including entering a state).

In the probability tier, these hypotheses will contain parts of product computed during Viterbi decoding, for example

\[
\text{PATHA} \quad b_1(o_1) b_2(o_2) b_3(o_3) b_4(o_4) \ldots
\]

\[
\text{PATHB} \quad A
\]

The inferences used will be as follows:

\[
\text{PATHA} + \text{[B]} \rightarrow \text{PATHB}
\]

\[
\text{PATHB} + \text{[A]} \rightarrow \text{PATHA}
\]

or exactly speaking with all the tiers they are:

\[
\begin{pmatrix}
t_1 & t_2 & t_3 \\
J_1 & J_2 & J_3 \\
\text{PATHA} \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
t_1 & t_2 & t_3 \\
J_1 & J_2 & J_3 \\
\text{PATHB} \\
\end{pmatrix}
\]

At the beginning of the recognition process we will express the transition matrix \( A \) as one hypothesis per nonzero matrix element and we will compute values of functions \( b_j(o) \) for all combinations of a speech segment \( o \) and an HMM state \( j \). We have thus two types of hypotheses:

\[
\begin{pmatrix}
t_x & t_y \\
J_1 & J_2 \\
\text{A} \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
t_1 & t_2 \\
J_1 & J_2 \\
\text{B} \\
\end{pmatrix}
\]

To get the type B hypotheses, we will store the observation vectors in hypotheses of type CEPSTRUM

\[
\begin{pmatrix}
2.00 & 2.02 \\
o_1 \text{CEPSTRUM} \\
2.02 & 2.04 \\
o_2 \text{CEPSTRUM} \\
2.04 & 2.06 \\
o_3 \text{CEPSTRUM} \\
\ldots 
\end{pmatrix}
\]

and use it as a questions for an author computing probabilities \( b() \). Module gets cepstrum \( o_i \) and sends back the value \( b_j(o_i) \) for all emitting states \( j \). The state identifier \( j \) is sent back in an answer together with the value \( b_j(o_i) \). Example communication is as follows:

\[
\begin{pmatrix}
2.00 & 2.02 \\
\text{CEPSTRUM} \\
\end{pmatrix} \equiv \begin{pmatrix} \text{B} \end{pmatrix} \equiv \begin{pmatrix}
2.00 & 2.02 \\
b_{e_2}(o_1) \\
\end{pmatrix}
\]
The module \( b() \) carries out usual computations of observation vector probabilities, given e.g. by Gaussian distributions:

\[
b_j(o) = g \cdot \exp\left(-\frac{1}{2} \sum_{k=1}^{n} \frac{(o_k - \mu_k)^2}{r_k}\right)
\]

where the constant \( g \) equals

\[
g = \frac{1}{\sqrt{(2\pi)^n \prod_{k=1}^{n} r_k}}
\]

and observation vectors, means and variances are

\[
o = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}, \quad r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}
\]

We can of course use multiple mixtures and multiple streams version of distribution; the exact form is not important here – it is implemented inside the module. Note that this is the only part of the Viterbi decoding which is not carried out by the inference engine itself.

To start the ‘reaction’ in the inference engine, we need the hypothesis

\[
\begin{array}{c}
2.00 \\
e_1 \\
\text{PATH A}
\end{array}
\]

and the Viterbi decoding begins:

\[
\begin{array}{c}
2.00 \\
e_1 \quad \text{e_1} \\
\text{PATH B}
\end{array} + \begin{array}{c}
T \\
T \\
0 \\
\text{PATH A}
\end{array} \rightarrow \begin{array}{c}
2.00 \\
\text{PATH A}
\end{array}
\]

\[
\begin{array}{c}
2.00 \\
e_1 \quad \text{e_2} \\
\text{PATH A}
\end{array} + \begin{array}{c}
\frac{a_{1,2}}{b_{1,2}(o_1)} \\
\frac{a_{1,2}}{B} \\
\text{PATH A}
\end{array} \rightarrow \begin{array}{c}
2.00 \\
\text{PATH A}
\end{array}
\]

After few steps, we will generate alternative competing paths corresponding to the same part of speech and ending in the same state. The better path will be chosen using coalescence:

\[
\begin{array}{c}
2.00 \\
\text{PATH A}
\end{array} + \begin{array}{c}
p_1 \\
\text{PATH A}
\end{array} \rightarrow \begin{array}{c}
2.00 \\
\text{PATH A}
\end{array}
\]

\[
\begin{array}{c}
2.00 \\
\text{PATH A}
\end{array} + \begin{array}{c}
p_2 \\
\text{PATH A}
\end{array} \rightarrow \begin{array}{c}
2.00 \\
\max(p_1, p_2) \\
\text{PATH A}
\end{array}
\]

4. PRONUNCIATION

Many languages has relatively regular pronunciation rules which can be expressed using context dependent grammar rules. Rules express relationship between local substrings of graphemes and substrings of phonemes. We can thus find possible written form of sub-word sized chunks of the utterance and use them to create partially known tier of the written form. Language model can be used on this partially known tier. This way it is possible to avoid generating large lists of possibilities on the lower levels of recognition - language model comes in the play very early.

Context dependent grammar rules can be easily expressed in our theory using two-tier hypotheses composed of phoneme tier and grapheme tier. Both tiers are context dependent strings. For example hypotheses

\[
\begin{array}{c}
b-e \\
\text{je} \\
\text{n+i} \\
\text{n}
\end{array}
\]

means that grapheme “e” is pronounced as “je” when preceded by grapheme “b” and grapheme “n” is pronounced “ni” when followed by grapheme “n”.

In these examples we used context dependency in the grapheme tier, however the inverse version is possible as well and we might even find automatic conversions between these two versions of pronunciation hypotheses sets.

5. LANGUAGE MODELS

For simple language models based on statistics of couples of adjacent words the realisation is very straightforward. These models have some value when used for English, they have however very low performance for languages using less strict ordering of words (made possible by more inflections and grammatical co-ordinations).

For Czech language, model based on grammatical co-ordination of possibly nonadjacent words might be used. The following technique is able to analyse the grammatical structure while still dealing only with continuous chunks of the utterance.

We will use special tier containing information about grammatical relation of words. For each word, this tier contains information about the relative position of another word which “controls” this one – it is either positive integer if the controlling word is on the right hand side, negative integer if it is on the left hand side, or question mark if it is at an unknown position (most probably on position out of the chunk). As an example, we will show possible course of anal-
ysis of the following sentence:

\[
\begin{array}{c}
\text{slon} & \text{snědl} & \text{zelený} & \text{ker}\ \\
1 & ? & 1 & -2
\end{array}
\]

Word “ker” is controlled by word “snědl” which is two words to the left from it (-2), word “zelený” is controlled by the word “ker” which is next to it on the right hand side (1) etc. At the beginning of the grammatical analysis we have only chunks containing individual words:

\[
\begin{array}{c}
\text{?} & \text{slon} & \text{?} & \text{snědl} & \text{?} & \text{zelený} & \text{?} & \text{ker} & \text{?}
\end{array}
\]

These chunks will be combined together; each combination glues two adjacent chunks together and replaces one questionmark by supposed relation between word in first chunk and word in second chunk. On the correct way of solution, following inferences are possible:

\[
\begin{array}{c}
\text{slon} & \text{?} & \text{snědl} & \text{?} & \text{?} & \rightarrow & \text{slon snědl} & \text{?} & \text{?} & \rightarrow & \text{1} & ? & 1 & -2
\end{array}
\]

\[
\begin{array}{c}
\text{?} & \text{zelený} & \text{?} & \text{ker} & \text{?} & \rightarrow & \text{zelený ker} & \text{?} & \text{?} & \rightarrow & \text{1} & ? & 1 & -2
\end{array}
\]

\[
\begin{array}{c}
\text{snědl} & \text{?} & \text{zelený ker} & \text{?} & \rightarrow & \text{snědl zelený ker} & \text{?} & \text{?} & \rightarrow & \text{1} & ? & 1 & -2
\end{array}
\]

\[
\begin{array}{c}
\text{slon} & \text{snědl} & \text{zelený ker} & \text{?} & \rightarrow & \text{slon snědl zelený ker} & \text{?} & \text{?} & \rightarrow & \text{1} & ? & 1 & -2
\end{array}
\]

\[
\begin{array}{c}
\text{slon snědl} & \text{zelený ker} & \text{?} & \rightarrow & \text{slon snědl zelený ker} & \text{?} & \text{?} & \rightarrow & \text{1} & ? & 1 & -2
\end{array}
\]

There are two ways leading to the right result – they differ in the order of grouping the chunks together; however the final result is the same:

\[
\begin{array}{c}
\text{slon snědl zelený ker} & \text{slon snědl zelený ker}
\end{array}
\]

Not every tree of grammatical coordination can be composed this way, however the excluded trees correspond to rather strange sentences like

\[
\begin{array}{c}
\text{čipce zelený snědl ker} \\
2 & ? & 2 & -2
\end{array}
\]

which might most likely be found in poetry only. In practice the limited number of possible combinations is desirable because it reduces number of possibilities to consider.

All we had to do to implement this parsing strategy was proper definition of the relational tier and its concatenation operation. In the above example we presented only the “right” hypotheses which lead to the right answer; in practice the concatenation operation has to generate more possible results and score them according to known statistics of grammatical coordination of individual couple of words (or classes of words). Statistics can be generated using big hand-analyzed text corpus, which is available for Czech.

6. PARALLEL PROCESSING

Continuous speech recognition requires huge processing power. Even if the final application is not supposed to run on a large machine, it is desirable to have enough processing power available during the system design phase when the software is not yet fully optimized. Recent advent of supercomputing on parallel machines built using ordinary personal computers, fast network interconnect and free operating system (so called 'beowulf' machines) offered unmatched price/performance ratio for many demanding tasks. Continuous speech recognition might also benefit from this new resource [2].

While the beowulf architecture is very attractive it is not always trivial to exploit its power. Software has to be parallelized by hand and its design must obey relatively low interconnect speed. On the other hand, using N computing nodes we combine not only N processors but also the cache and RAM sizes. Well designed software can exploit this larger possibility to run in-RAM (without trashing) and in-cache and achieve superlinear speedup much higher than N. Structures resulting from the use of our theory in recognition system design are well suited for this desired modus operandi.

7. CONCLUSION

This article covers brief overview of our approach. Interested reader will find much more details on our web server (http://noel.feld.cvut.cz/hypo). Software implementing the theory will be released at EURO-SPEECH'99 starting day under the terms of the GPL so it will be available free of charge.

REFERENCES