Sprint
The RWTH Speech Recognition System

David Rybach

rybach@i6.informatik.rwth-aachen.de

22. November 2007

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

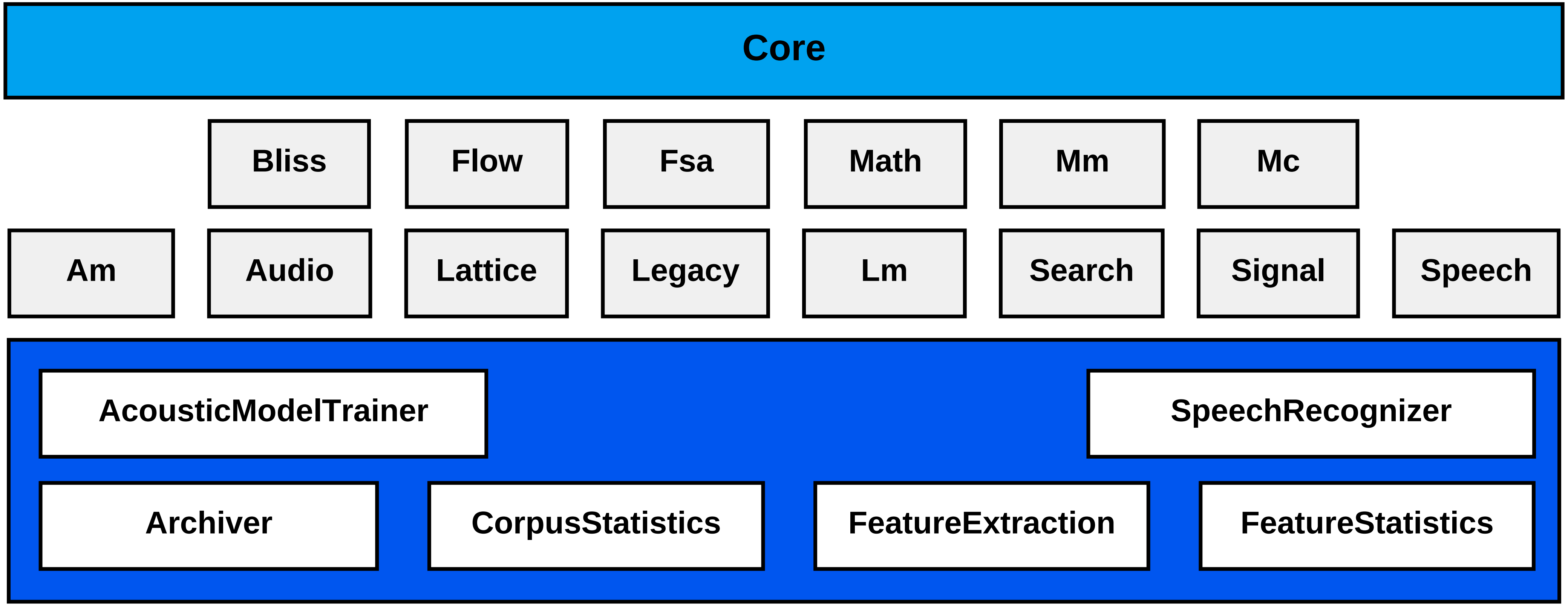
Outline

- **Software Overview**
- **Recognition**
- **Outlook: Training**

Overview: Sprint

- Development started 2001 by Stephan Kanthak and Max Bisani
- Further development by several PhD students at i6
- Today: standard system for all ASR research topics and projects
- Very flexible and extendable
- Framework also used for machine translation, video / image processing

Overview: Modules



Overview: Modules

Basic Modules

Core	I/O, configuration, parser, utilities
Bliss	"Better Lexical Information Sub-System": lexicon, corpus
Flow	general data processing framework, used mainly for feature extraction
Fsa	finite-state automata library [Kanthak & Ney 04] (separately available)
Math	math library, interface to LAPACK
Mm	mixture models
Mc	model combination

Modules depending on basic modules

Am	acoustic modeling: HMM, state-tying, adaptation
Audio	audio data input (Flow nodes)
Lattice	lattice processing
Legacy	"old" stuff: decision tree
Lm	language modeling
Search	search algorithms, alignment generator, state tree
Signal	signal processing for feature extraction
Speech	higher level algorithms for speech processing

Configuration

- Components receive their configuration from a global configuration instance
- The configuration is set up from configuration files and command line arguments
- A configuration resource is a key-value pair
- Keys have the from `<selector1>. ... .<selectorN>.<parameter>`
- Each selector corresponds to a component name
- Components are organized hierarchically
- Selectors can be replaced by the wildcard `*`
- Components choose the configuration resource with the best matching (most specific) selectors
- Precedence: configuration files, command line
- Example:

```
*.corpus.file                            = recognition.corpus
*.acoustic-model.hmm.states-per-phone = 3
```

```
speech-recognizer.linux-intel-standard --config=recognition.config \
                                       --*.corpus.file=other.corpus
```

Configuration

- Configuration resources with equal selectors can be grouped
- Example:

```
[*.acoustic-model.hmm]
states-per-phone                = 3
state-repetitions               = 2
```

- Configuration values can include **references**: `$(selector)`
- The reference is textually replaced by the resolved value
- Example:

```
DESCRIPTION = epps-eval07-es.pass-2
SGE_TASK_ID = 0004
lattice-archive.path            = data/$(DESCRIPTION).lattices.$(SGE_TASK_ID)
```

- Configuration files can include other files using the `include` directive
- Example:

```
include data/shared.config
```

Channel

- Sprint components use channels for messages / text output
- The content written to a channel is sent to its targets
- Each channel can be configured and redirected separately
- Configuration of channels and targets is done using the standard configuration process
- Channel targets can be either predefined targets (`stdout`, `stderr`, `nil`) or custom targets defined by the configuration

```
[*]
log.channel                  = output-channel
warning.channel              = output-channel, stderr
error.channel                = output-channel, stderr
dot.channel                  = nil

recognizer.statistics.channel = output-channel, recognizer_stat

[*.channels.output-channel]
file              = log/my-logfile.log
append            = false
encoding          = UTF-8
unbuffered        = true
compressed        = false
```

Corpus

The corpus file defines

- audio files
- segmentation
- unique identifiers for segments and recordings
- audio / segment meta data (speaker, recording condition, ...)
- transcriptions (for training / simple WER computation)

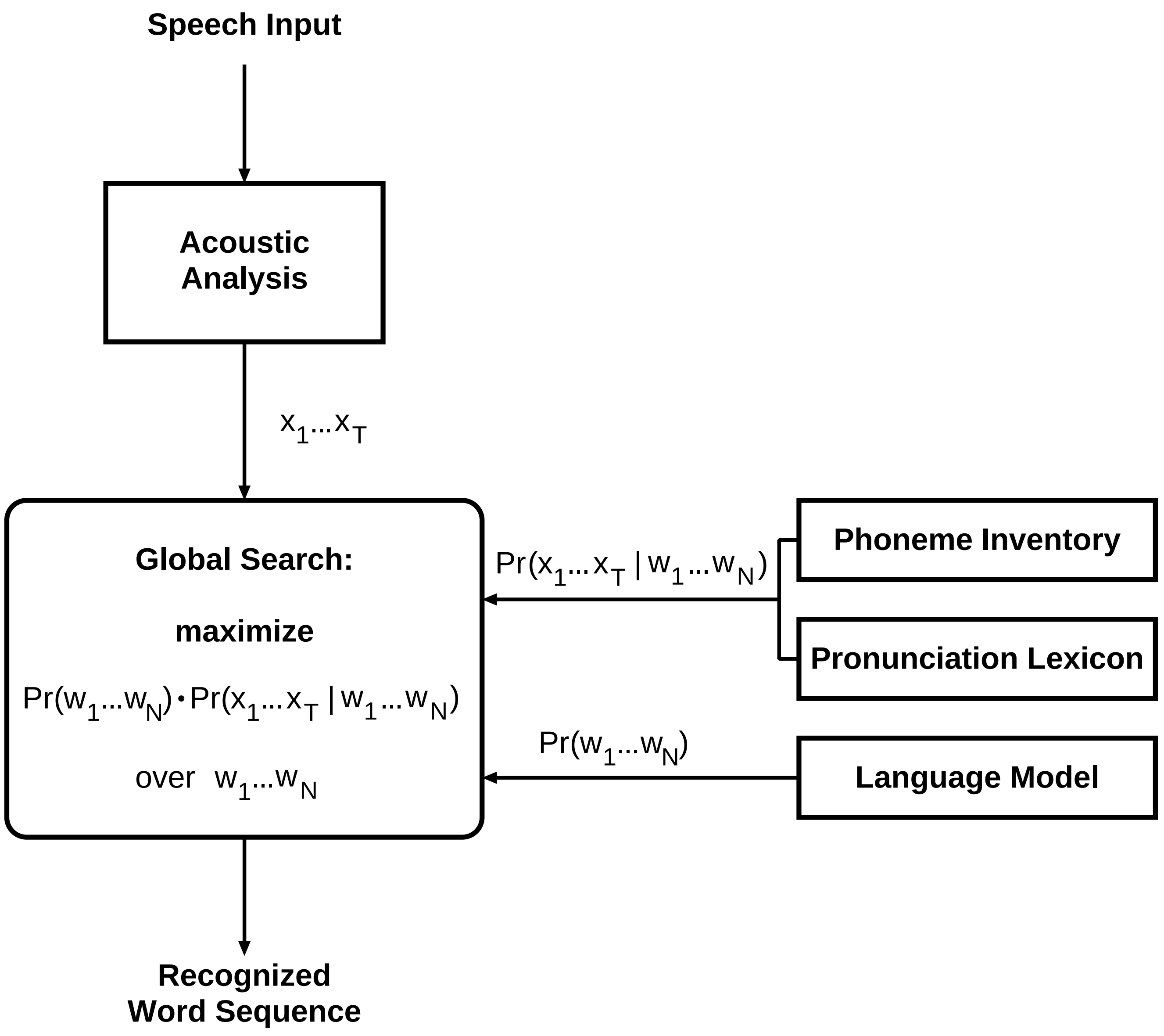
```
<corpus name="TC-STAR_ES">
    <speaker-description name="spk1">
        <dialect> native </dialect>
        <name> VIDAL-QUADRAS ROCA, Alejo </name>
        <accent></accent>
        <id>spk1</id>
        <type>male</type>
    </speaker-description>
    <recording audio="20050127/1000_1055_ES_SAT.wav" name="20050127_1000_1055_ES_SAT">
        &lt;segment start="287.393" end="290.192" name="287.393-290.192">
            <speaker name="spk1"/>
            <orth>
                Señorías ocupen sus asientos .
            </orth>
        &lt;/segment>
    </recording>
</corpus>
```

Corpus

- Most applications have a corpus driven architecture:
  Sequence of processed files is defined by the corpus
- Parallelization: split the task by corpus partitioning
- Configuration:
  - `corpus.partition`: number of partitions
  - `corpus.select-partition`: partition used by the actual process

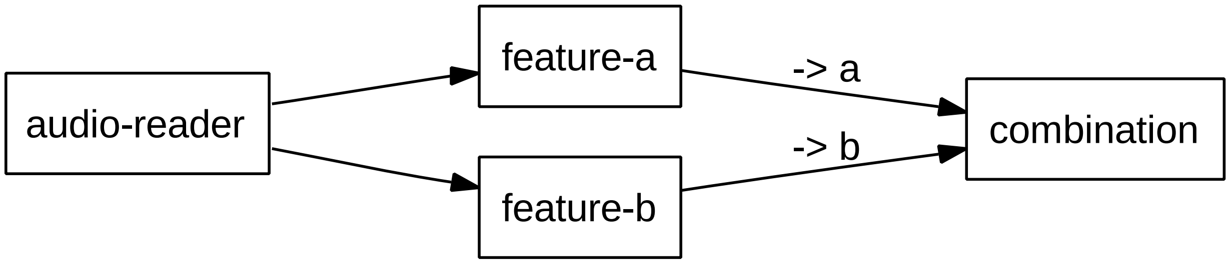
- Main classes:
  - `Bliss::CorpusDescription`
  - `Bliss::CorpusVisitor` (cf. Visitor Pattern)

Recognition: Overview



Feature Extraction: Flow

- **Flow is a general framework for data processing**
- **Currently it is mainly used for feature extraction and alignment generation**
- **Data manipulation (including loading / storing) is done by nodes**
- **A node has zero or more input and output ports**
- **Nodes are connected by links from one port to another**
- **A set of nodes combined by links form a network**
- **A network can be used as node itself**
- **Flow networks are defined in XML documents**
- **A node is either a piece of code inside Sprint or a network defined by another flow file**
- **Abstract Example:**



Feature Extraction: Flow

```
<network name="lda"> <!-- file: lda.flow -->
  <in name="in"/>
  <out name="out"/>

  <node name="sliding-window" filter="signal-vector-f32-sequence-concatenation"/>
  <link from="lda:in" to="sliding-window"/>
  <node name="multiplication" filter="signal-matrix-multiplication-f32" file="$(lda-file)"/>
  <link from="sliding-window" to="multiplication"/>
  <link from="multiplication" to="lda:out"/>
</network>
```

```
<network>
  <out name="features"/>
  <param name="input-file"/>
  <param name="start-time"/>
  <param name="end-time"/>

  <node name="audio-reader" filter="audio-input-file-$(audio-format)
        file="$(input-file)" start-time="$(start-time)" end-time="$(end-time)"/>

  <node name="feature-a" filter="some-node"/>
  <link from="audio-reader" to="feature-a"/>

  <node name="lda" filter="lda.flow"/>
  <link from="feature-a" to="lda:in"/>

  <link from="lda:out" to="network:features"/>
</network>
```

Feature Extraction: Flow Caches

- All data sent in a flow network can be written / loaded from caches
- A cache node has one input and one output port
- Data requested at the output port of a cache node:
  - first check if the data is present in the cache
  - if not, the data is requested at the node's input port (if connected)
  - and stored in the cache
- Cached items are identified by the parameter `id`
- Segment ids are propagated by the network
- Caches are used primary to store features and alignments

```
<node name="base-feature-extraction" filter="$(file)"/>

<node name="base-feature-cache" filter="generic-cache" id="$(id)"/>
<link from="base-feature-extraction" to="base-feature-cache"/>

<node name="lda" filter="lda.flow"/>
<link from="base-feature-extraction-cache" to="lda:in"/>
```

Feature Extraction: MFCC + Voicedness

RWTH TC-STAR EPPS system:

- 16 MFCCs
- Segment-wise cepstral mean normalization
- Voicedness feature [Zolnay & Schlüter^{+} 02]
- MFCCs and single voicedness are augmented.

Flow files:

- `mfcc/pre-filterbank.flow`:
  preemphasis, Hamming window, FFT, amplitude spectrum
- `mfcc/filterbank.flow`: Mel frequency warping, critical band integration
- `mfcc/post-filterbank.postprocessing.flow`:
  logarithm, cosine transform
- `mfcc/postprocessing.flow`: mean normalization
- `mfcc/mfcc.postprocessing.flow`: combining network
- `voicedness/*.flow`: computation of voicedness feature

Feature Extraction: VTLN

Vocal tract length normalization:

- Warping of conventional Mel warped filter-bank. [Welling & Kanthak^{+} 99]
- Target models of warping factor estimation: single Gaussian
- Fast VTLN: classify warping factors online

Implementation using Flow:

```
<!-- vtln/warped-mfcc.recognized-factors.postprocessing.flow -->
...
<node name="unwarped-features" .../>
...
<node name="warping-factor-recognizer" filter="signal-bayes-classification"/>
<link from="unwarped-features" to="warping-factor-recognizer:feature-score-weight"/>
<link from="unwarped-features:target" to="warping-factor-recognizer"/>

<node name="warped-filterbank"  filter="signal-filterbank" filter-width="268.258"
    warping-function="nest(linear-2($input(warping-factor), 0.875), mel)"/>
<link from="unwarped-features" to="warped-filterbank"/>
<link from="warping-factor-recognizer" to="warped-filterbank:warping-factor"/>
...
```

Alternative: static mapping from segment names to warping factors

Feature Extraction: VTLN

Flow files:

- `concatenations/vtln.voicedness.flow`
- `vtln/warped-mfcc.recognized-factors.postprocessing.flow`

Configuration (`*.feature-extraction`)

- `linear-transform.file`:
  file name of the total scatter matrix used for variance normalization of the features
- `warping-factor-recognizer.class-label-file`:
  warping factor class labels
- `warping-factor-recognizer.likelihood-function.file`:
  models for the warping factor classifier

Feature Extraction: LDA

- Concatenate 9 frames in a sliding window (153 coefficients) [Haeb-Umbach & Ney 92]
- Project to 45 dim. output

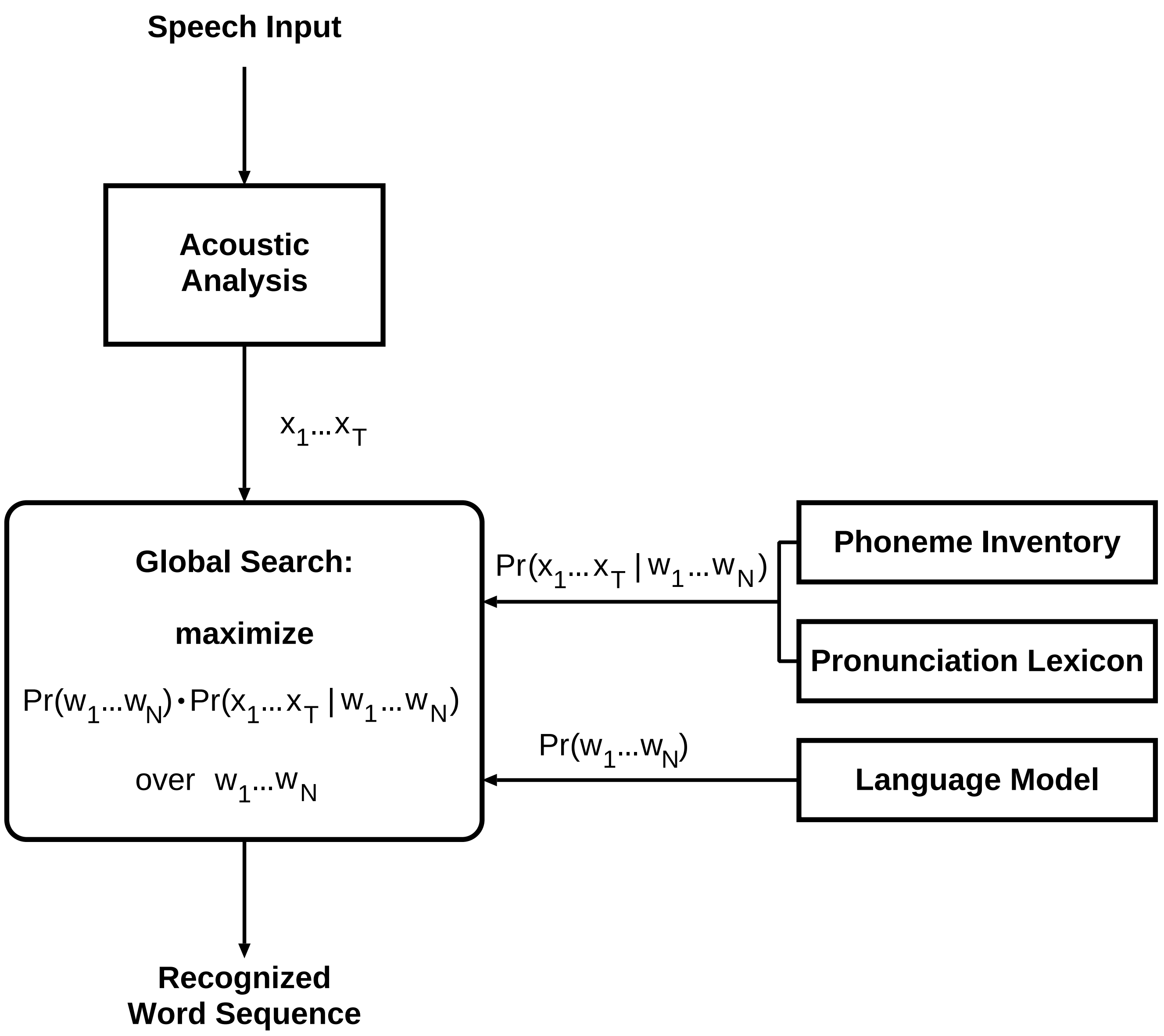
Implementation using Flow:

```
<!-- shared/lda-window.flow -->
<node name="lda-window" filter="signal-vector-f32-sequence-concatenation"
      max-size="9" right="4" margin-condition="present-not-empty" expand-timestamp="false"/>

<!-- shared/lda.flow -->
<node name="multiplication" filter="signal-matrix-multiplication-f32"
      file="$(file)"/>
<link from="lda-window" to="multiplication"/>
```

Configuration:

- `lda.file`: file name of the projection matrix
- `lda-window.max-size`: number of frames concatenated
- `lda-window.right`: offset to center the sliding window


Speech Input
Acoustic Analysis
x1...xT
Global Search:
maximize
Pr(w1...wN)•Pr(x1...xT | w1...wN)
over w1...wN
Pr(x1...xT | w1...wN)
Phoneme Inventory
Pronunciation Lexicon
Pr(w1...wN)
Language Model
Recognized Word Sequence

Lexicon

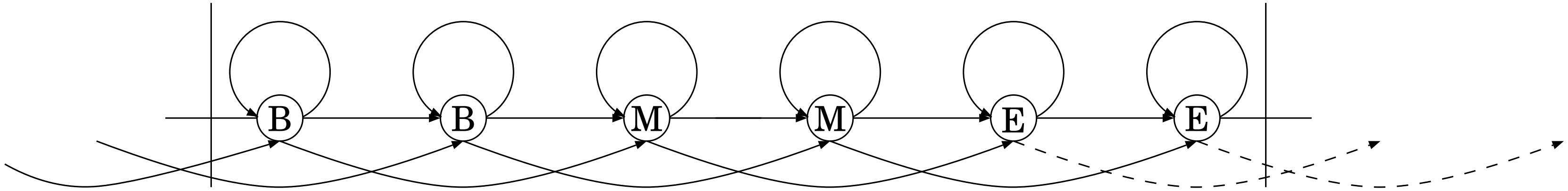
- The Lexicon defines the phoneme set and the pronunciation dictionary
- The basic unit of the lexicon is a lemma
- A lemma can consist of
  - one or more orthographic forms (written word)
  - any number of pronunciations (phonetic transcriptions)
  - a sequence of syntactic tokens (language model tokens)
  - a sequence of evaluation tokens (word used for evaluation)
- Special lemmata define properties of silence, sentence begin/end, unknown words

- Example: RWTH TC-STAR EPPS lexicon for Spanish:
  - 38 phonemes
  - 61,031 lemmas
  - 61,959 pronunciations

```xml
<lexicon>
    <phoneme-inventory>
        <phoneme>
            <symbol>a</symbol>
        </phoneme>
        <phoneme>
            <symbol>B</symbol>
        </phoneme>
        <!-- ... -->
        <phoneme>
            <symbol>si</symbol>
            <variation>none</variation> <!-- no context dependency -->
        </phoneme>
    </phoneme-inventory>
    <lemma special="silence">
        <orth>[SILENCE]</orth>         <!-- transcription -->
        <orth>[pause]</orth>
        <phon score="0.0">si</phon>     <!-- phonemes + pronuncation score -->
        <synt/>                         <!-- language model token          -->
        <eval/>                         <!-- word used for evaluation      -->
    </lemma>
    <lemma>
        <orth>Al-Qaeda</orth>
        <phon score="0.558977109356">a l k a e D a</phon>
        <phon score="0.69314718056">a l si k a e D a</phon>
    </lemma>
    <lemma>
        <orth>Chechenia</orth>
        <phon score="0.0">tS e tS e n j a</phon>
    </lemma>
</lexicon>
```

Acoustic Model: HMM

- Common HMM topology for one phoneme: 3 states with state repetitions:

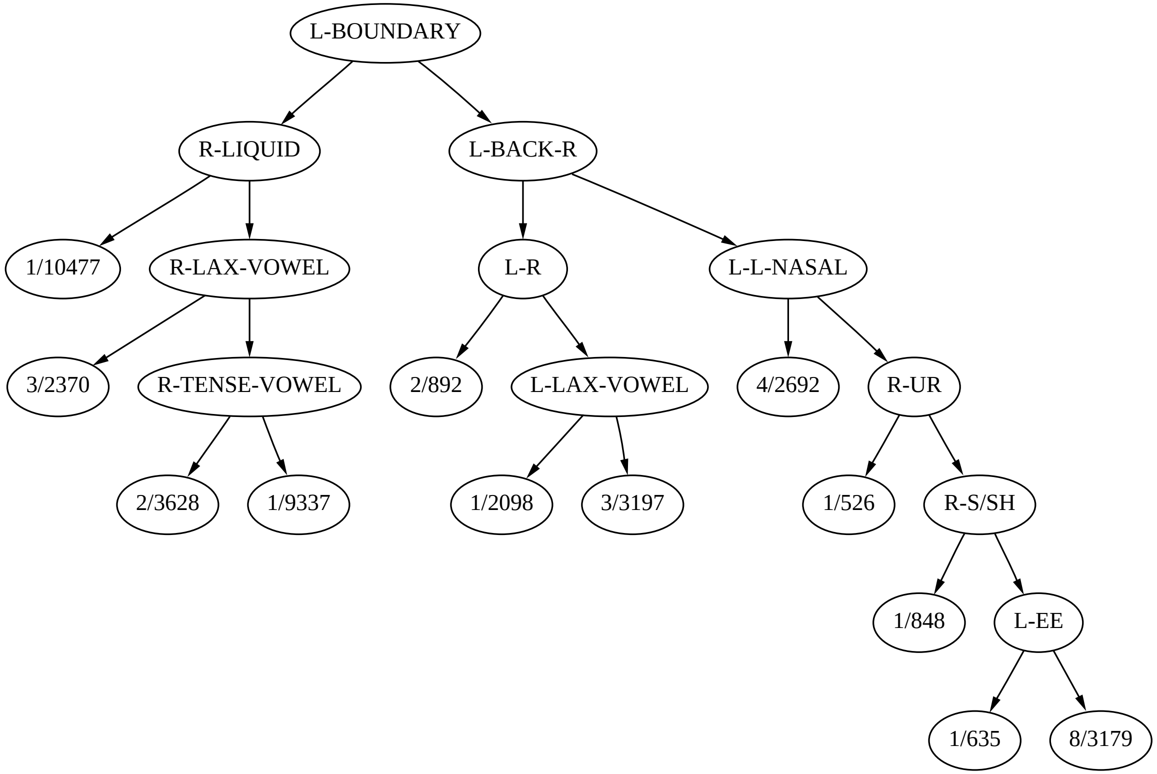


- Parameters: `hmm.states-per-phone`, `hmm.state-repetitions`
- Transition probabilities are defined state independently for loop, forward, skip, and exit transitions (`tdp`)
- The silence phoneme has only 1 state (`tdp.silence`)
- RWTH systems use phonemes in triphone context (larger context is possible in principle, but not tested) (`phonology`)
- Across-word context dependency is used in most systems (`across-word-model`)
- Main classes
  `Am::ClassicStateModel`, `Am::ClassHmmTopology`, `Blis::Phonology`

Acoustic Model: State Tying

Classification and Regression Tree (CART)

- **Tie parameters of "similar" triphones / allophones using a phonetic decision tree [Beulen & Bransch^{+} 97]**
- **RWTH TC-STAR EPPS Systems: 37,071 triphones in the recognition lexicon $\rightarrow$ 4500 generalized triphone states**
- **Configuration:** `acoustic-model.decision-tree`

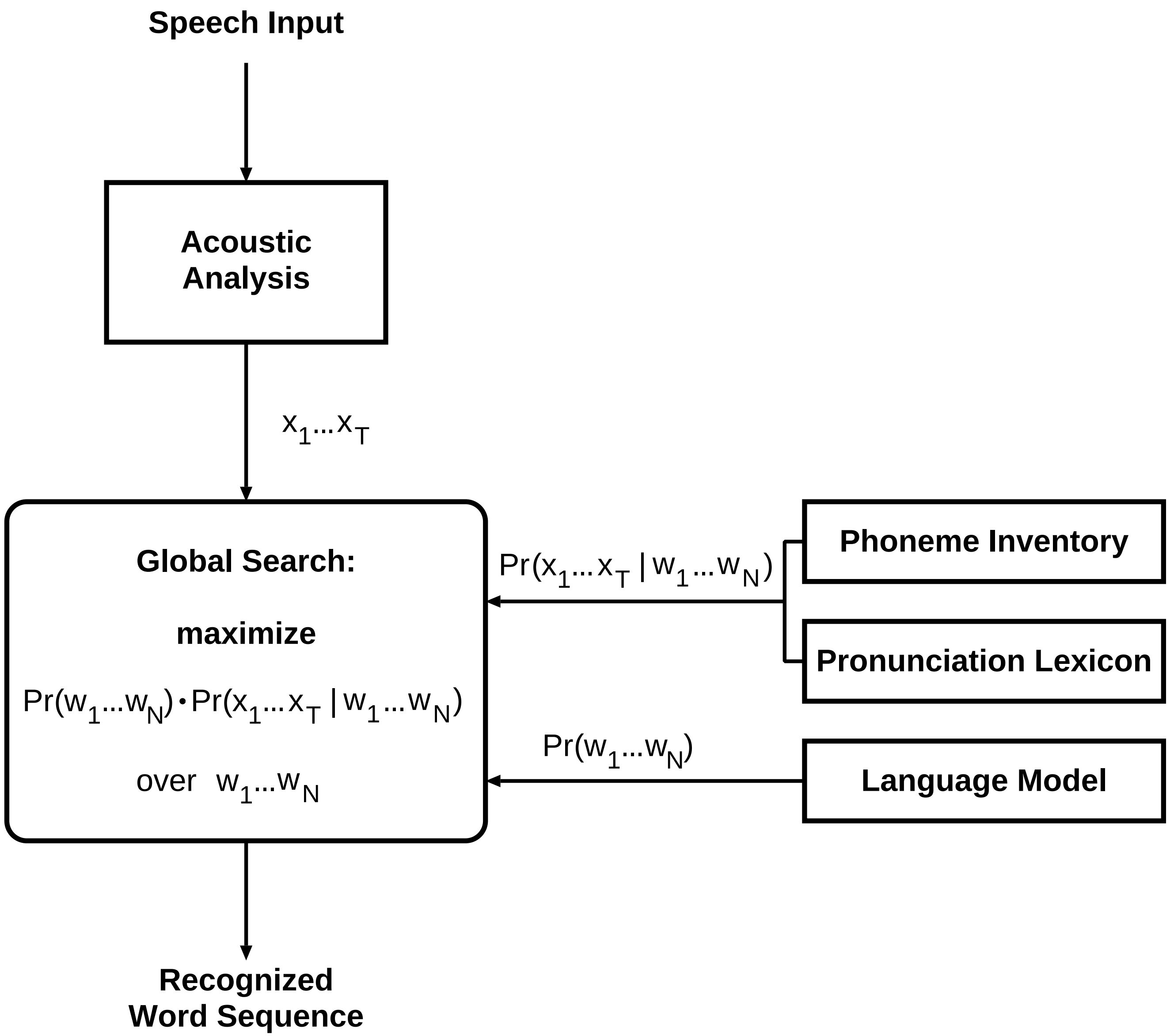


Acoustic Model: Mixture Model

- The emission probabilities of the HMMs are modelled by Gaussian mixture distributions
- RWTH TC-STAR models consist of
  872,386 (es) / 899,552 (en) densities in 4501 mixtures
  with a globally pooled diagonal covariance matrix
- Mixture models are stored as estimators (sums and counts) and estimated after loading
- The mixture model is used by a feature scorer
- Common feature scorer for recognition: `SIMD-diagonal-maximum`
  using quantized features and SIMD instructions (MMX/SSE2)
  [Kanthak & Schütz^{+} 00]
- Configuration: `acoustic-model.mixture-set`
- Main classes:
  `Mm::MixtureSet`, `Mm::GaussDensity`, `Mm::MixtureSetEstimator`

Language Model

- Commonly used: backing off language models in ARPA LM format
- Training using the the SRI Language Modeling Toolkit (SRILM)
- RWTH TC-STAR EPPS systems:
  - 4-gram with 14.3M multi-grams (es) / 7.5M (en)
  - Smoothing: modified Kneser-Ney discounting with interpolation
- Language model can be stored in a binary representation (image) to speed up loading
- Configuration (`lm`)
  - `type`: language model type (`ARPA`, `zerogram`, ...)
  - `file`: language model file
  - `image`: image file
- Main classes:
  `Lm::BackingOffLm`, `Lm::ArpaLm`


Speech Input
Acoustic Analysis
$x_1 \ldots x_T$
Global Search:
maximize
$Pr(w_1 \ldots w_N) \cdot Pr(x_1 \ldots x_T \mid w_1 \ldots w_N)$
over $w_1 \ldots w_N$
$Pr(x_1 \ldots x_T \mid w_1 \ldots w_N)$
Phoneme Inventory
Pronunciation Lexicon
$Pr(w_1 \ldots w_N)$
Language Model
Recognized Word Sequence

Search

- Across-word model search [Sixtus & Ney 02]
- Tree lexicon (state tree) can be computed in advance (`state-tree.file`)
- Recognition results and statistics are saved in log files
- Processing of log files using Analog
- Basic evaluation of recognition results using Analog
- Advanced evaluation by conversion into NIST ctm format

Search: example recognition log file

```
<recording name="20060906_1500_1815_OR_SAT" full-name="EPPS/20060906_1500_1815_OR_SAT" ...
  &lt;segment name="4980.615-4985.285" full-name="EPPS/20060906_1500_1815_OR_SAT/4980.615-4985.285"
    <orth source="reference">
      le pero puede ser un ensayo de éxito para
    </orth>
    <traceback>
      t=0       s=0         #|#
      t=25      s=1298.11   [B]                  /Gint/    #|#
      t=27      s=1413.78   [SILENCE]                 /si/    #|#
      t=52      s=3111.29   le                  /l e/    e|p
      t=107     s=6897.59   pero                  /p e r o/    #|#
      t=112     s=7274.36   [ARTIC]                  /Gspe/    #|#
      t=122     s=8039.06   [ARTIC]                  /Gspe/    #|#
      t=152     s=9922.66   puede                  /p w e D e/    e|s
      t=177     s=11463     ser                  /s e r/    #|#
      t=195     s=12562.4   un                  /u n/    n|e
      t=277     s=17467.6   ensayo                  /e n s a jj o/    #|#
      t=292     s=18539.5   de                  /d e/    #|#
      t=377     s=23949.8   éxito                  /e G s i t o/    #|#
      t=439     s=27559.6   para                  /p a r a/    #|#
      t=466     s=28981.5   [B]                  /Gint/    #|#
      t=467     s=29119.4   #|#
    </traceback>
    <orth source="recognized">
      [B] [SILENCE] le pero [ARTIC] [ARTIC] puede ser un ensayo de éxito para [B]
    </orth>
   <search-space-statistics>
     <statistic name="trees before pruning" type="scalar">
     ....
```

Search: Model Combination

$$\arg\max_{w_1^N} \left\{ \alpha \sum_{n=1}^{N} \log\, p(w_n | w_{n-2}^{n-1}) \right.$$
$$\left. + \max_{s_1^T} \sum_{t=1}^{T} \left[\beta \log\, p(s_t | s_{t-1}, w_1^N) + \gamma \log\, p(x_t | s_t, w_1^N) \right] \right\}$$

- α **language model scale: `lm.scale`**
- β **transition probability scale: `acoustic-model.tdp.scale`**
- γ **acoustic model scale: `acoustic-model.mixture-set.scale`**

- **pronuncation scale: weight the pronuncation scores** **`model-combination.pronunciation-scale`**

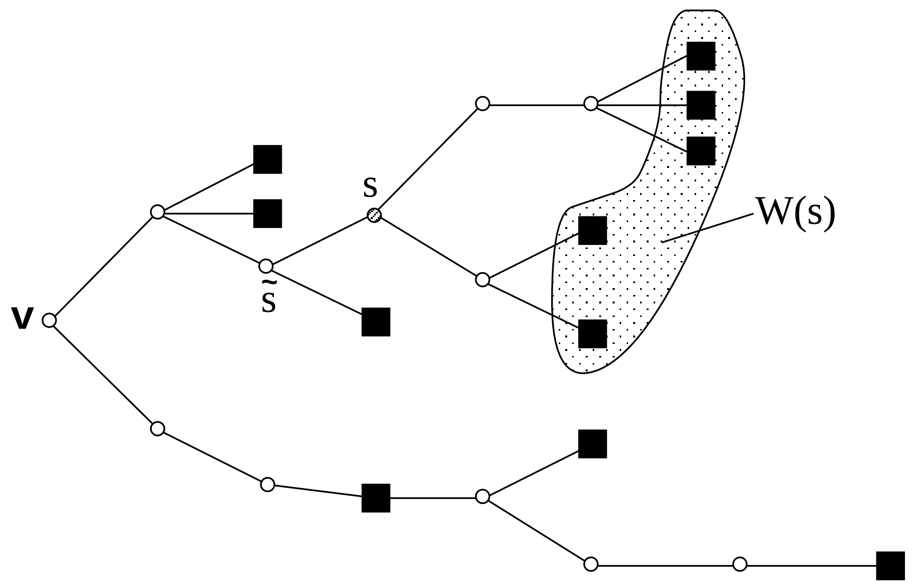
Search: Pruning

- **Acoustic pruning**
  - determine score of best hypothesis at each time t: $Q_{AC}(t)$
  - eliminate hypothesis if its score is lower than $f_{AC} \cdot Q_{AC}(t)$
  - `acoustic-pruning`: $\log(f_A C)$
  - `acoustic-pruning-limit`: limit the number of active hypotheses

- **Language model pruning**
  - Prune tree roots after language model recombination
  - `lm-pruning`: difference between word end scores to best word end score at each time t
  - `lm-pruning-limit`: limit number of active word end hypotheses

Search: LM Lookahead

- Idea: use the knowledge from the language model as early as possible in the search process.
- From a certain state only a small subset of leaves i.e. word ends can be reached.
- Use the best possible score to refine acoustic pruning
- Configuration (`lm-lookahead`)
  - `history-limit`: length of history considered for LM look-ahead
  - `tree-cutoff`: maximum depth of state tree covered by LM look-ahead



Multipass Recognition

RWTH TC-STAR recognition schema:

- (1. pass + segmentation)
- 2. pass: speaker independent models
- segment clustering
- CMLLR estimation
- 3. pass recognition
- MLLR estimation
- 4. pass recognition

Segment Clustering

- Bottom-up clustering using the Bayesian Information Criterion (BIC) as stop criterion [Chen & Gopalakrishnan 98]
- Segment clusters are used as target classes for adaptation techniques
- The clustering itself is performed on the acoustic features (usually plain MFCC features)
- Flow file: `segment-clustering/segment-clustering.flow`
- Configuration (`segment-clustering`)
  - `file`: resulting cluster file
  - `lambda`: penalty weight, penalizes the complexity of the model
  - `mincluster` / `maxcluster`: limit the number of clusters

Alignment

- Alignment from features to allophone states
- The alignment is implemented in a Flow node

```
<network>
  <out name="features"/>
  <out name="alignments"/>
  <!-- .... -->
  <param name="id"/>
  <param name="orthography"/>

  <node name="feature-extraction-setup" filter="$(file)"
        input-file="$(input-file)"
        id="$(id)" start-time="$(start-time)" end-time="$(end-time)" track="$(track)"/>
  <link from="feature-extraction-setup:features" to="network:features"/>

  <node name="aggregate" filter="generic-aggregation-vector-f32"/>
  <link from="feature-extraction-setup:features" to="aggregate"/>

  <node name="alignment" filter="speech-alignment"
        id="$(id)" orthography="$(orthography)"/>
  <link from="aggregate" to="alignment"/>

  <node name="alignment-cache" filter="generic-cache"
        id="$(id)"/>
  <link from="alignment" to="alignment-cache"/>
  <link from="alignment-cache" to="network:alignments"/>
```

Speaker Adaptation: CMLLR

- Constrained MLLR (“feature space MLLR”)
- Single feature transform per speaker/cluster
- Estimated on single Gaussian: simple target model [G. Stemmer 05]
- In recognition: unsupervised adaptation. Use preliminary recognition output

Usage in Sprint:

- Create an alignment on the previous recognition output using a single Gaussian model (“split 0 mixture”)
- Estimate CMLLR transformation matrices using the alignment and the segment clusters
- Adapted recognition using the transformation matrices in the feature extraction process

Speaker Adaptation: MLLR

- Unsupervised maximum likelihood linear regression mean adaptation
- Estimation implementation uses Viterbi approximation (in frame state alignment),
  maximum approximation (on the Gaussian component level)
- Regression classes are implemented using a regression class tree
- Commonly used: pruned CART from the acoustic model
- RWTH TC-STAR EPPS systems: 5,000 observations per class,
  1,000 observations for the silence class
- Usually used in combination with CMLLR

Usage in Sprint

- Alignment generation
- Estimation of transformations
- Adapted recognition using the transformation matrices to adapt
  the acoustic model

Outlook: Training

- Define / generate lexicon
- Acoustic model training steps;
  - alignment (features to allophone states)
  - CART estimation
  - LDA estimation
  - mixture model estimation
    - accumulation: collect feature vectors for each mixture
    - estimation: estimate mixture model
    - splitting: split mixtures
  - estimate VTLN models
  - estimate CMLLR transforms for speaker adaptive training
  - testing, tuning, iterating, ...
- Main training tool: `AcousticModelTrainer`
- LM-Training: external tools (SRILM)

Thank you for your attention

David Rybach

`rybach@i6.informatik.rwth-aachen.de`

`http://www-i6.informatik.rwth-aachen.de/`

Appendix

References

[Beulen & Bransch$^+$ 97] K. Beulen, E. Bransch, H. Ney: State-Tying for Context Dependent Phoneme Models. In *Proc. Fifth Europ. Conf. on Speech Communication and Technology*, Vol. 3, pp. 1179–1182, Rhodes, Greece, Sept. 1997. 23

[Chen & Gopalakrishnan 98] S. Chen, P.S. Gopalakrishnan: Clustering via the Bayesian information criterion with applications in speech recognition, pp. 645–648, 1998. 33

[G. Stemmer 05] D.G. G. Stemmer, F. Brugnara: Adaptive Training Using Simple Target Models. Vol. 1, pp. 997 – 1000, Philadelphia, PA, USA, March 2005. 35

[Haeb-Umbach & Ney 92] R. Haeb-Umbach, H. Ney: Linear Discriminant Analysis for Improved Large Vacabulary Continuous Speech Recognition, 1316, March 1992. 18

[Kanthak & Ney 04] S. Kanthak, H. Ney: FSA: An Efficient and Flexible C++ Toolkit for Finite State Automata Using On-Demand Computation. In *Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, pp. 510–517, Barcelona, Spain, July 2004. 5

[Kanthak & Schütz+ 00] S. Kanthak, K. Schütz, H. Ney: Using SIMD Instructions For Fast Likelihood Calculation in LVCSR. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2000)*, pp. 1531–1534, Istanbul, Turkey, June 2000. 24

[Sixtus & Ney 02] A. Sixtus, H. Ney: From Within-Word Model Search to Across-Word Model Search in Large Vocabulary Continuous Speech Recognition. *Computer Speech and Language*, Vol. 16, No. 2, pp. 245–271, May 2002. 27

[Welling & Kanthak+ 99] L. Welling, S. Kanthak, H. Ney: Improved Methods for Vocal Tract Normalization. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1999)*, pp. 761–764, Phoenix, Arizona, USA, March 1999. 16

[Zolnay & Schlüter+ 02] A. Zolnay, R. Schlüter, H. Ney: Robust Speech Recognition using a Voiced-Unvoiced Feature. In *Proc. Int. Conf. on Spoken Language Processing*, Vol. 2, pp. 1065–1068, Denver, CO, USA, Sept. 2002. 15