Graph Search and Lattices in ASR

CS 224S / LINGUIST 285 Spoken Language Processing

May 5, 2022

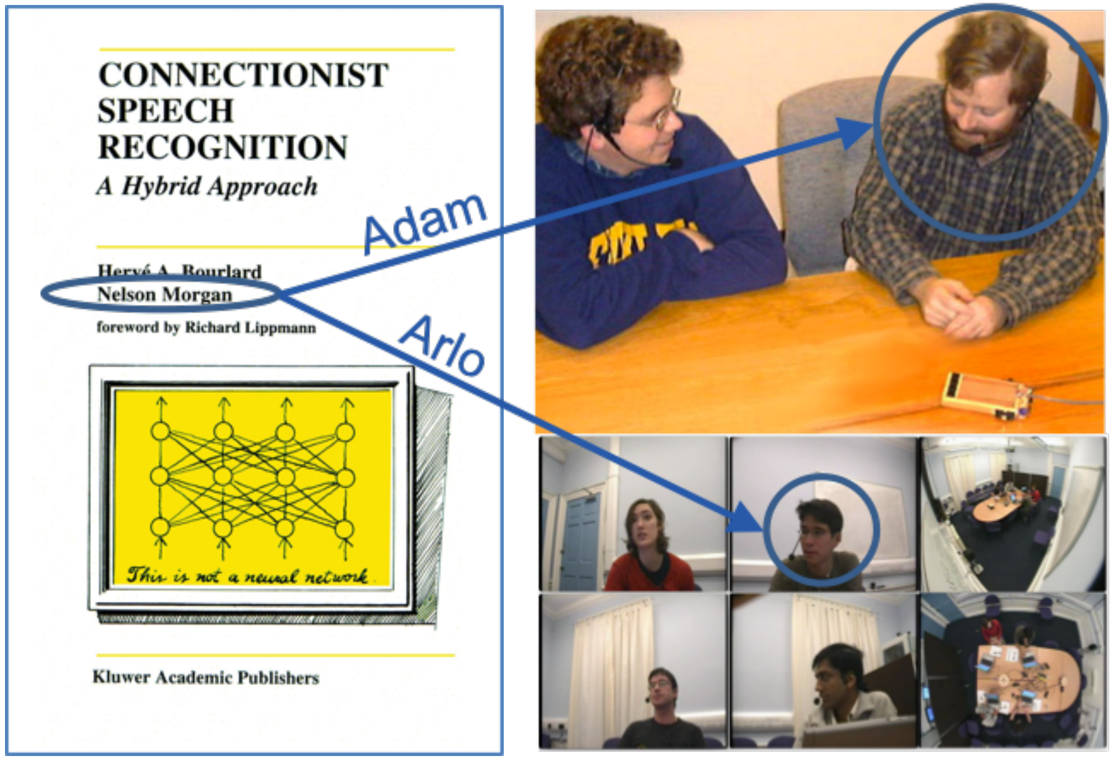
Guest Lecturer: Arlo Faria

arlo@mod9.com

Background: ICSI research → spinoff

©1994

2000 - 2006



2007 - 2018

Global Autonomous Language Exploitation

calo

The Parallel Computing Laboratory

2019 - 2022

Stanford NLP

MOD9

ASR Engine

©2019-2022 Mod9 Technologies v. 1.9.2

Motivation: graph search and lattices

Data is limited; customization is necessary.

Theory: models should be interpretable ~~fine-tuned~~.

Application: DIY functionality; not professional services.

Errors are inevitable; mitigation is necessary.

Theory: represent ~~recognize~~ what might be ~~is~~ spoken.

Application: search and editing; not captions or dictation.

Conclusion: use WFST ~~E2E~~ framework (i.e. Kaldi).

Kaldi: extensible HMM-DNN toolkit

Dan Povey: HMM-GMM → WFST-DNN → K2-FSA

Code structure

`egs/`	Scripts to train and evaluate systems.
`src/`	C++ libraries and Unix-style binaries.
`tools/`	Dependencies: OpenFST and BLAS.

Private fork

`egs/`	Use `fisher_swbd` recipe; add 15,000 hours of data.
`src/`	Modify I/O; add server w/ graph & lattice functions.
`tools/`	Add TensorFlow, Boost, SRC, VAD, etc.

Kaldi: modern approach to HMM

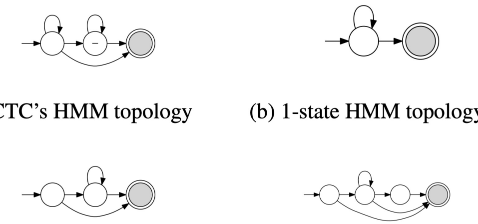
Structure: CTC-like model

Context: left biphones

Transition probabilities = 0.5

Training: lattice-free MMI

Features: 40d MFCC @ 30ms step



(a) CTC's HMM topology (b) 1-state HMM topology (c) 2-state HMM topology (d) 3-state HMM topology

Figure 1: *Different HMM topologies. The state marked with "-" is CTC's blank state and is shared across all the labels.*

After comparing various topologies, we settled on a topology where the first frame of a phone has a different label than the remaining frames (a different pdf-id, in Kaldi terminology, i.e. it maps to a different output of the neural net), so a single HMM may emit either a, or ab, or abb, etc. The reader is free to consider the b as analogous to the blank symbol in CTC (while bearing in mind that in general each triphone may get its own version of the b symbol).

We build the phonetic-context decision tree specifically for this topology and frame rate after converting alignments from a traditional HMM-GMM system at the normal frame rate; the decision-tree is then built using the same procedure and the same features (MFCC+LDA+MLLT) as for our HMM-GMM system. The optimal number of leaves tends to be a little smaller than than for a cross-entropy neural network.

2.2. Transition modeling

In our baseline cross-entropy based HMM-DNN framework, the HMMs use transition probabilities; these are estimated in the conventional way for HMMs. In this work we just set the transition probabilities to be a constant value (0.5) that makes each HMM-state sum to one. For the topologies we use, estimating the transition probabilities would add no modeling power anyway (depending on the exact granularity with which they are shared).

Figure 1: danielpovey.com/files/2018_interspeech_end2end.pdf

Text: danielpovey.com/files/2016_interspeech_mmi.pdf

Kaldi: practical approach to DNN

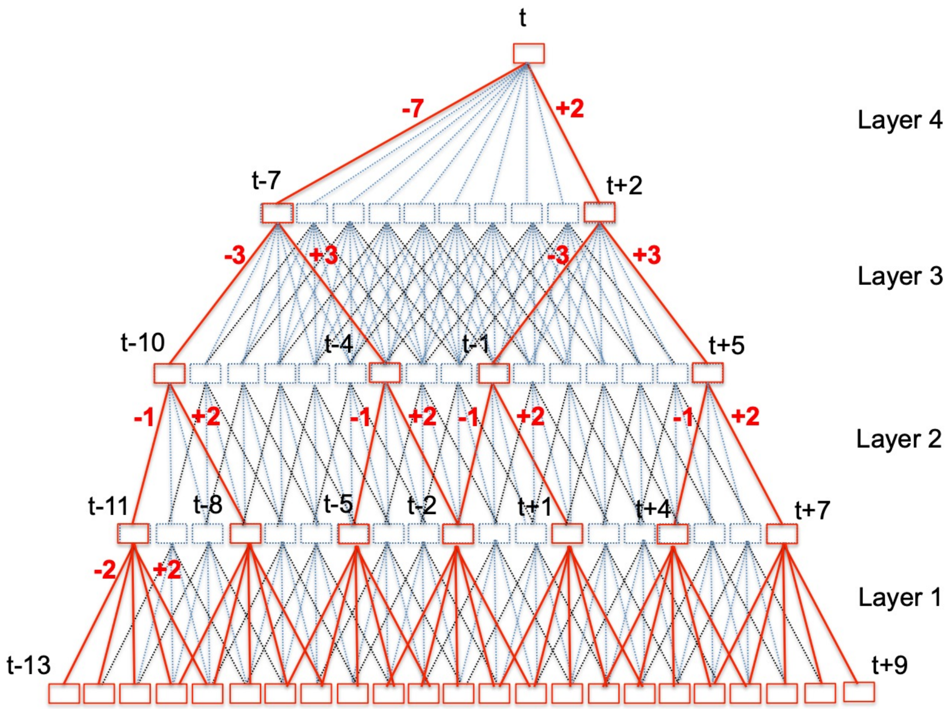


Figure 1: Computation in TDNN with sub-sampling (red)

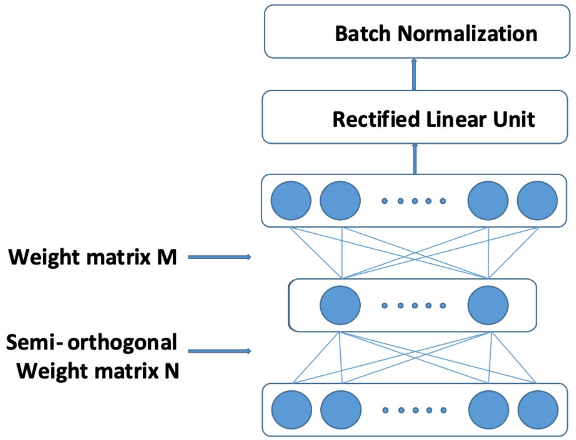


Figure 2: *Factorized layer with semi-orthogonal constraint*

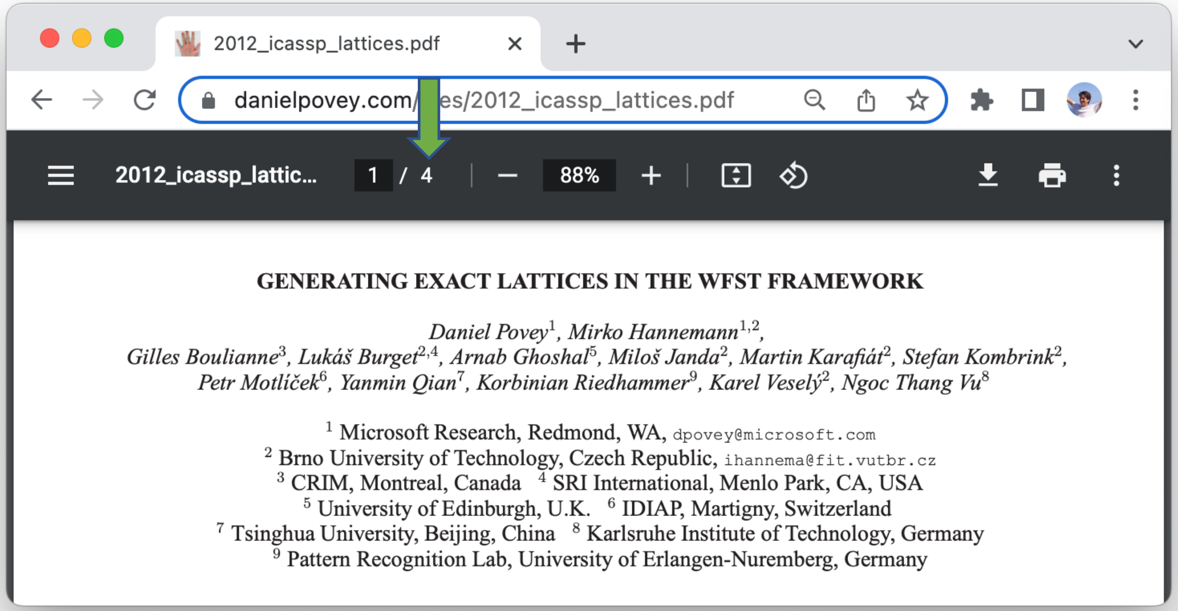
Table 1: *WER for TDNN models on Switchboard LVCSR task.*

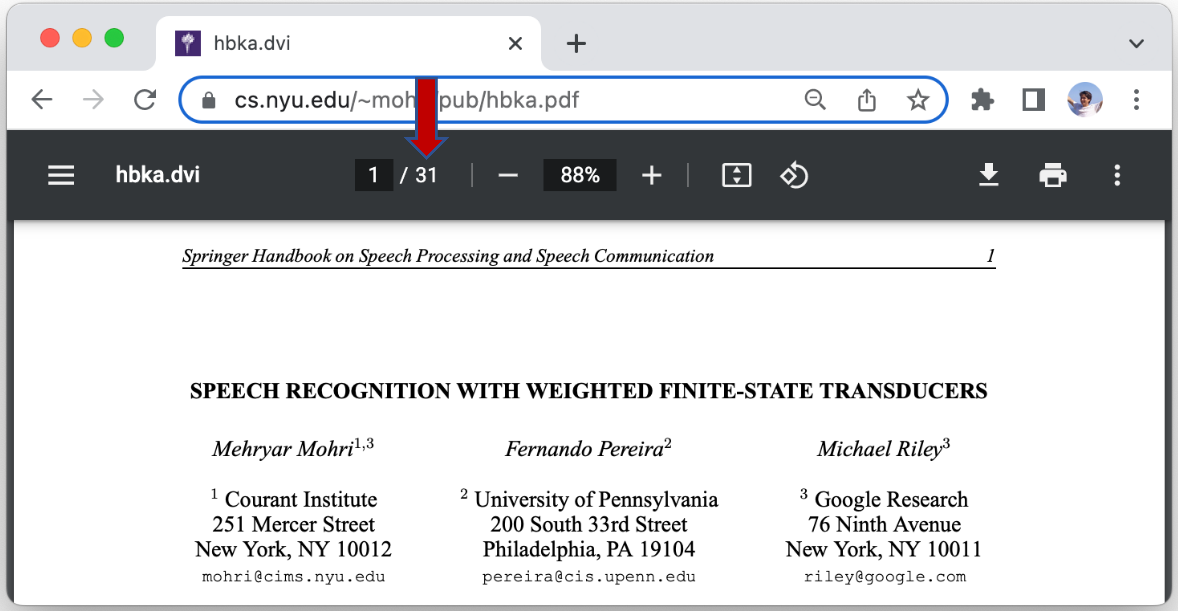
Acoustic Model	Size	Eval2000		RT03	Time [4](s)
		SWBD	Total		
Baseline TDNN (625)	19M	9.5	14.3	17.5	90
+ l2 regularization		9.1	14.0	16.9	96
Baseline TDNN (1536)	80M	9.4	14.6	17.2	211
+ l2 regularization		9.0	13.9	16.6	210
Factorized TDNN (1536-256)	20M	9.7	14.4	17.4	154
+ l2 regularization		9.1	13.9	17.0	155
++ semi-orthogonal		9.2	**13.7**	**16.0**	147

Figure 1: danielpovey.com/files/2015_interspeech_multisplice.pdf

Figure 2, Table 1: danielpovey.com/files/2018_interspeech_tdnnf.pdf

Reading: recommended vs. optional





Reading: recommended vs. optional

The graph creation process we use in our toolkit, Kaldi [1], is very close to the standard recipe described in [2], where the Weighted Finite State Transducer (WFST) decoding graph is

$$HCLG = \min(\det(H \circ C \circ L \circ G)), \qquad (1)$$

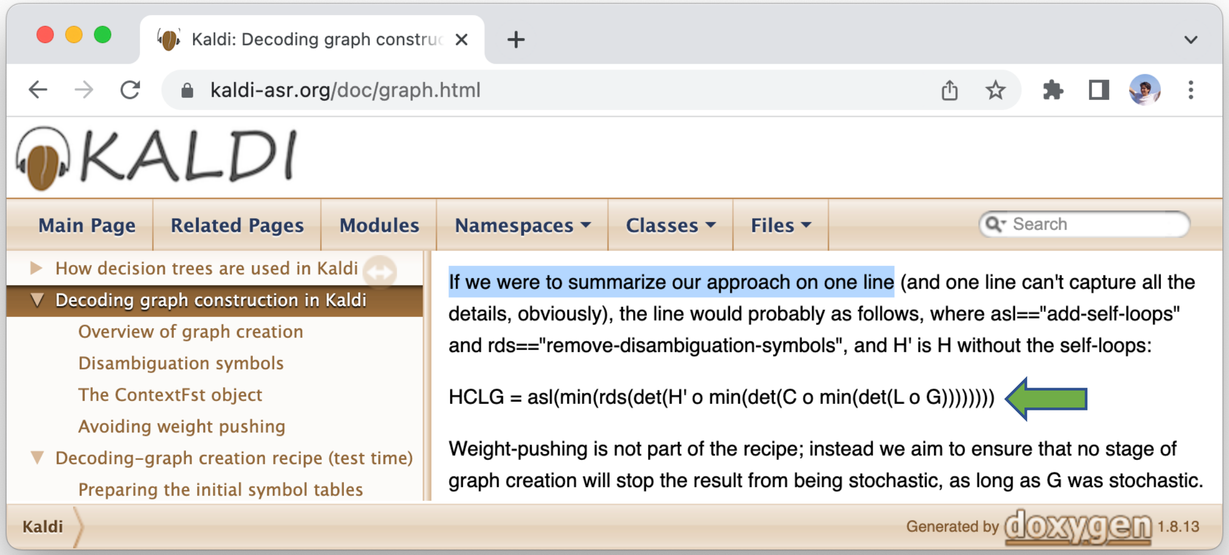


Table 1: *Semiring examples.* $\oplus_{\log}$ *is defined by:* $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

SEMIRING	SET	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

ply to countable sums (Lehmann [1977] and Mohri [2002] give precise definitions). The Boolean and tropical semirings are closed, while the probability and log semirings are not.

A *weighted finite-state transducer* $T = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ over a semiring $\mathbb{K}$ is specified by a finite input alphabet $\mathcal{A}$, a finite output alphabet $\mathcal{B}$, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a finite set of transitions $E \subseteq Q \times (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{B} \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an initial state weight assignment $\lambda : I \to \mathbb{K}$, and a final state weight assignment $\rho : F \to \mathbb{K}$. $E[q]$ denotes the set of transitions leaving state $q \in Q$. $|T|$ denotes the sum of the number of states and transitions of T.

Weighted automata (or weighted acceptors) are defined in a similar way by simply omitting the input or output labels. The *projection* operations $\Pi_1(T)$ and $\Pi_2(T)$ obtain a weighted automaton from a weighted transducer T by omitting respectively the input or the output labels of T.

Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is a sequence of consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. The path π is a *cycle* if $p[e_1] = n[e_k]$. An *ϵ-cycle* is a cycle in which the input and output labels of all transitions are ϵ.

closed, this is defined even for infinite R. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$. For an acceptor, we denote by $P(q, x, q')$ the set of paths with input label x. These definitions can be extended to subsets $R, R' \subseteq Q$ by $P(R, R') = \cup_{q \in R,\, q' \in R'} P(q, q')$, $P(R, x, y, R') = \cup_{q \in R,\, q' \in R'} P(q, x, y, q')$, and, for an acceptor, $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. A transducer T is *regulated* if the weight associated by T to any pair of input-output strings (x, y), given by

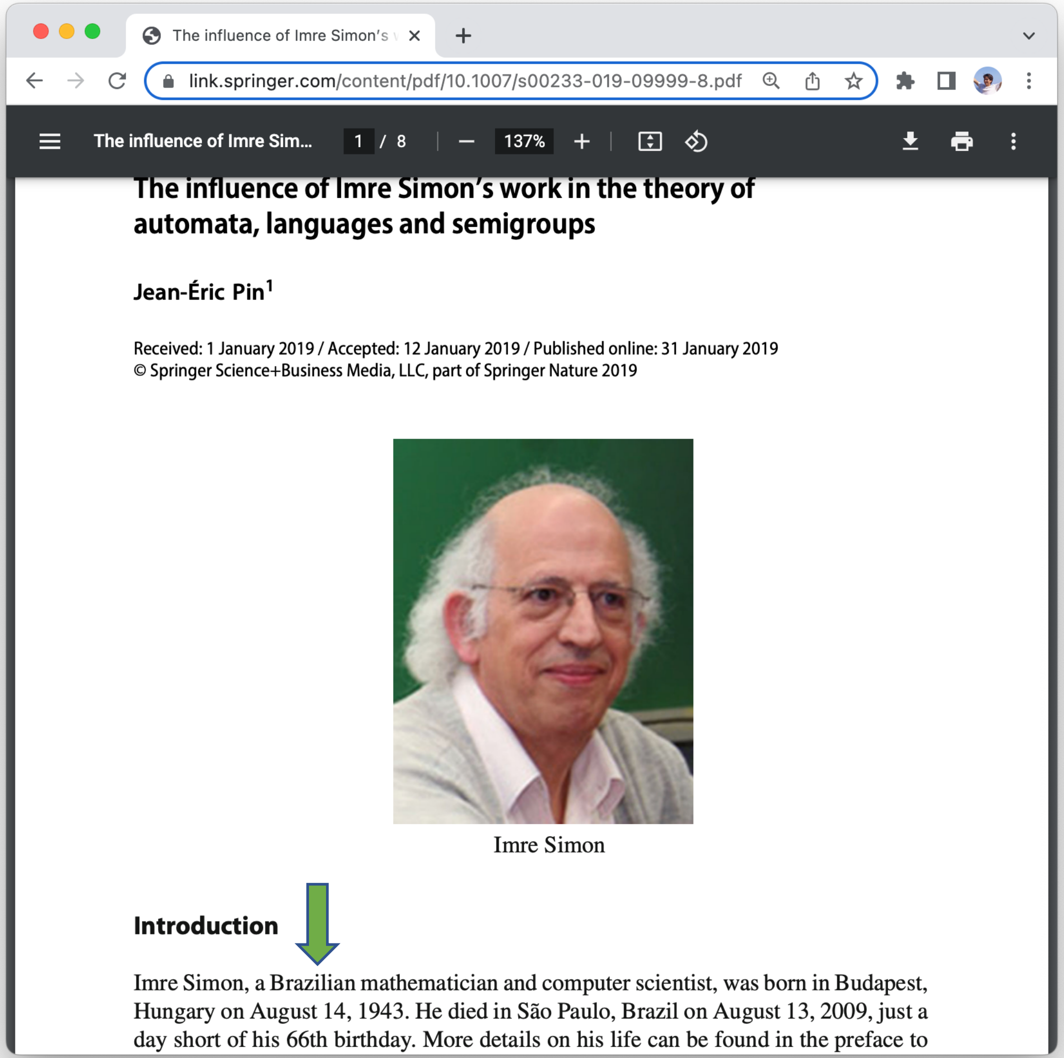
$$T(x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]], \quad (9)$$

is well defined and in $\mathbb{K}$. If $P(I, x, y, F) = \emptyset$, then $T(x, y) = \overline{0}$. A weighted transducer without ϵ-cycles is regulated, as is any weighted transducer over a closed semiring. Similarly, for a regulated acceptor, we define

$$T(x) = \bigoplus_{\pi \in P(I, x, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]]. \quad (10)$$

The transducer T is *trim* if every state occurs in some path $\pi \in P(I, F)$. In other words, a trim transducer has no useless states. The same definition applies to acceptors.

Tropical semiring?



The influence of Imre Simon's work in the theory of automata, languages and semigroups

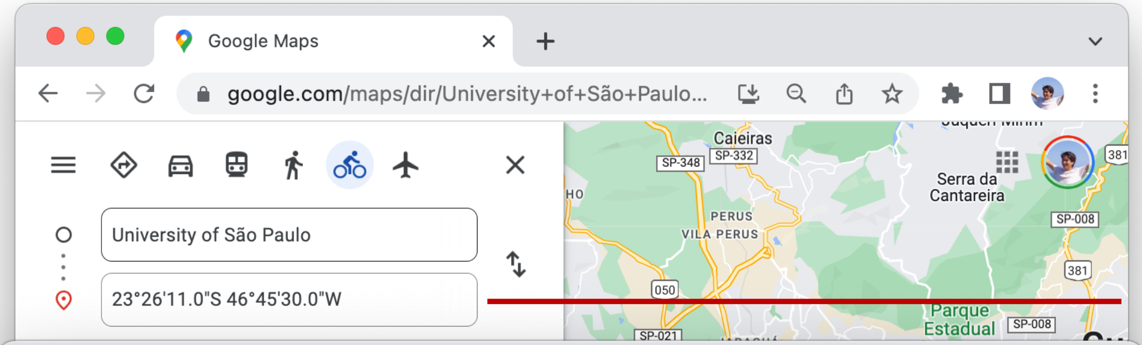
Jean-Éric Pin[1]

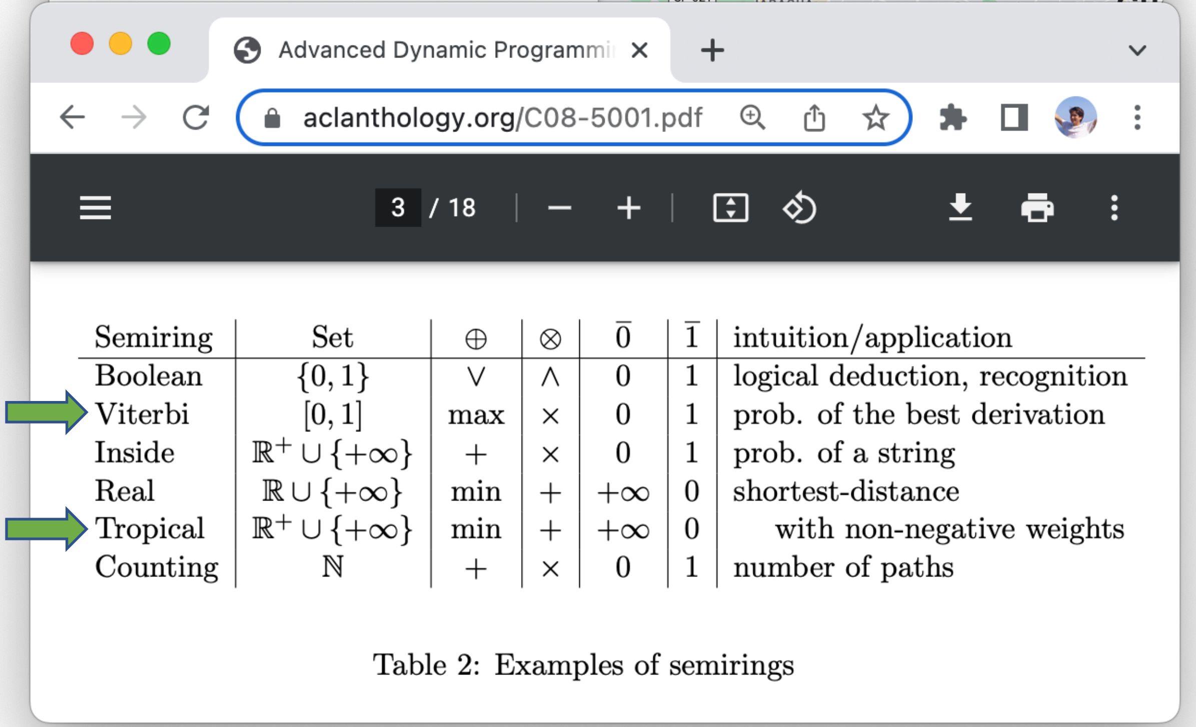
Received: 1 January 2019 / Accepted: 12 January 2019 / Published online: 31 January 2019

Imre Simon

Introduction

Imre Simon, a Brazilian mathematician and computer scientist, was born in Budapest, Hungary on August 14, 1943. He died in São Paulo, Brazil on August 13, 2009, just a day short of his 66th birthday. More details on his life can be found in the preface to



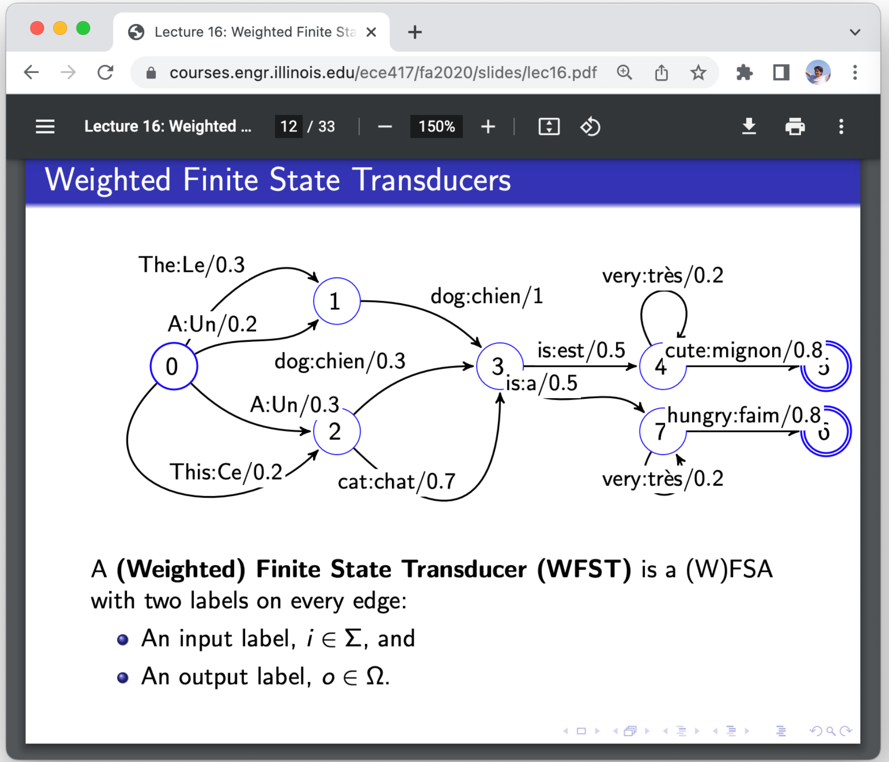


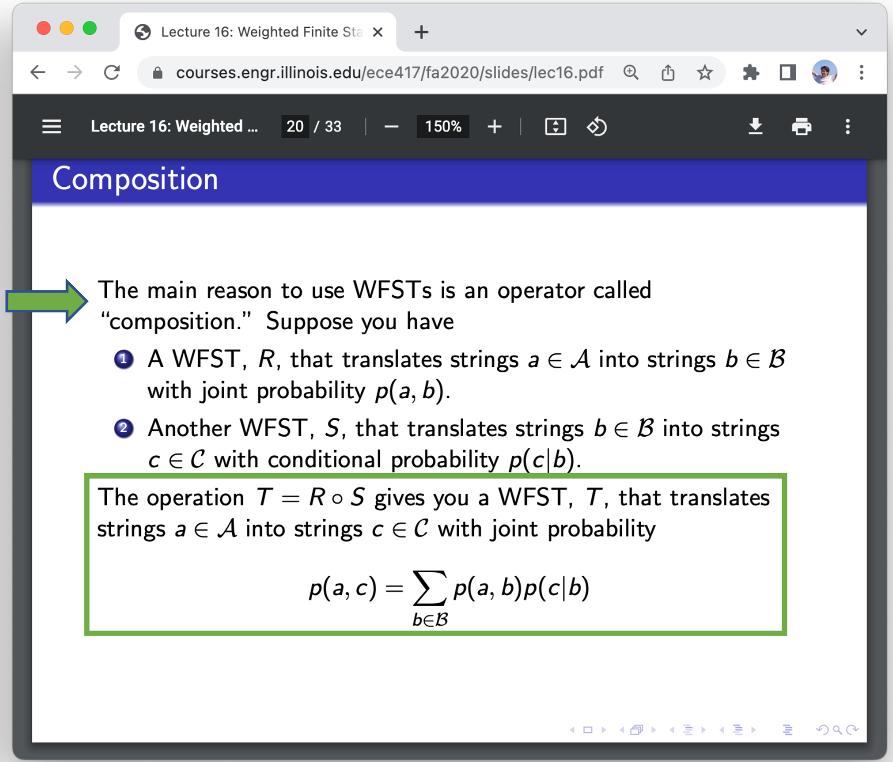
Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$	intuition/application
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1	logical deduction, recognition
Viterbi	$[0,1]$	max	$\times$	0	1	prob. of the best derivation
Inside	$\mathbb{R}^+ \cup \{+\infty\}$	$+$	$\times$	0	1	prob. of a string
Real	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$	0	shortest-distance
Tropical	$\mathbb{R}^+ \cup \{+\infty\}$	min	$+$	$+\infty$	0	with non-negative weights
Counting	$\mathbb{N}$	$+$	$\times$	0	1	number of paths

Table 2: Examples of semirings

W?FS[AT]

Recommend: awnihannun.com/writing/automata_ml.html
Optional: openfst.org/twiki/bin/view/FST/FstBackground
Helpful: courses.engr.illinois.edu/ece417/fa2020/slides/lec16.pdf





WFST operations

1. **Composition**
   Intuitive in theory; may be deferred in practice.
   Kaldi: static graph (huge) or dynamic lookahead (slow).

2. **Determinization, minimization, ε-removal, etc.**
   Complex optimizations, in theory and practice.
   Kaldi: specialized algorithms, beyond OpenFST.

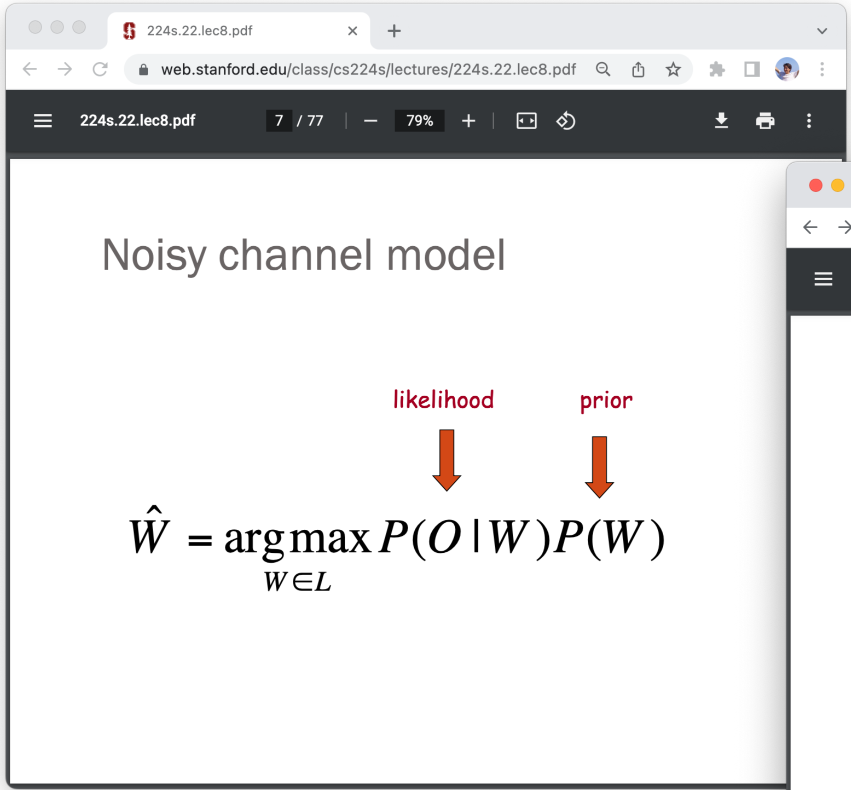
3. **Best path**
   Intuitive in theory; may be pruned in practice.
   Kaldi: decode to lattices … and rescore from lattices!

Previously in CS224S …

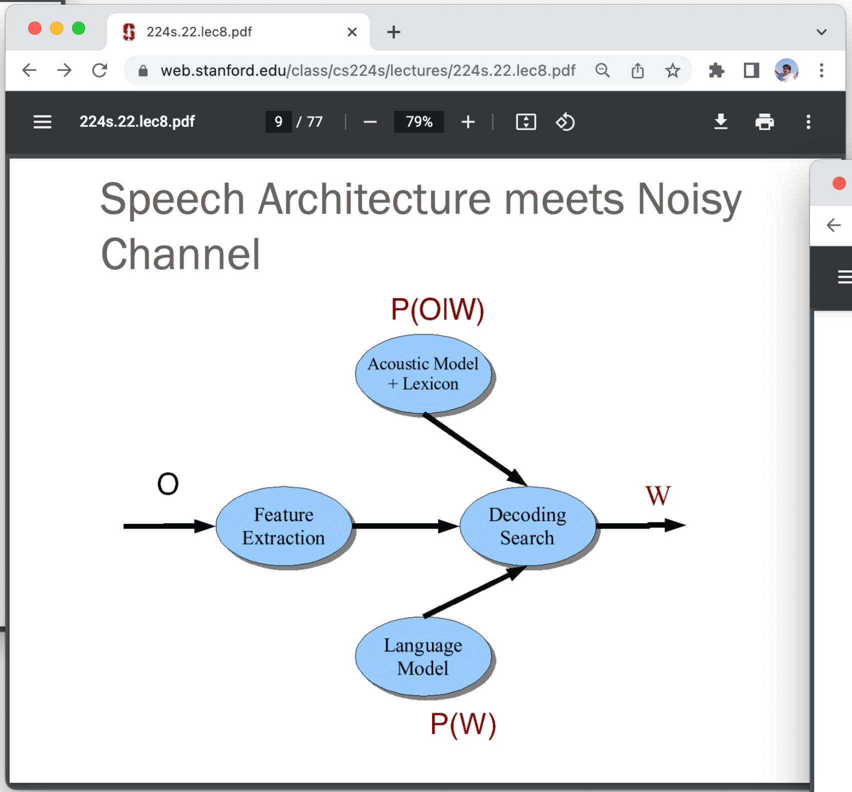
danielpovey.com/files/2012_icassp_lattices.pdf

$$S \equiv U \circ HCLG$$

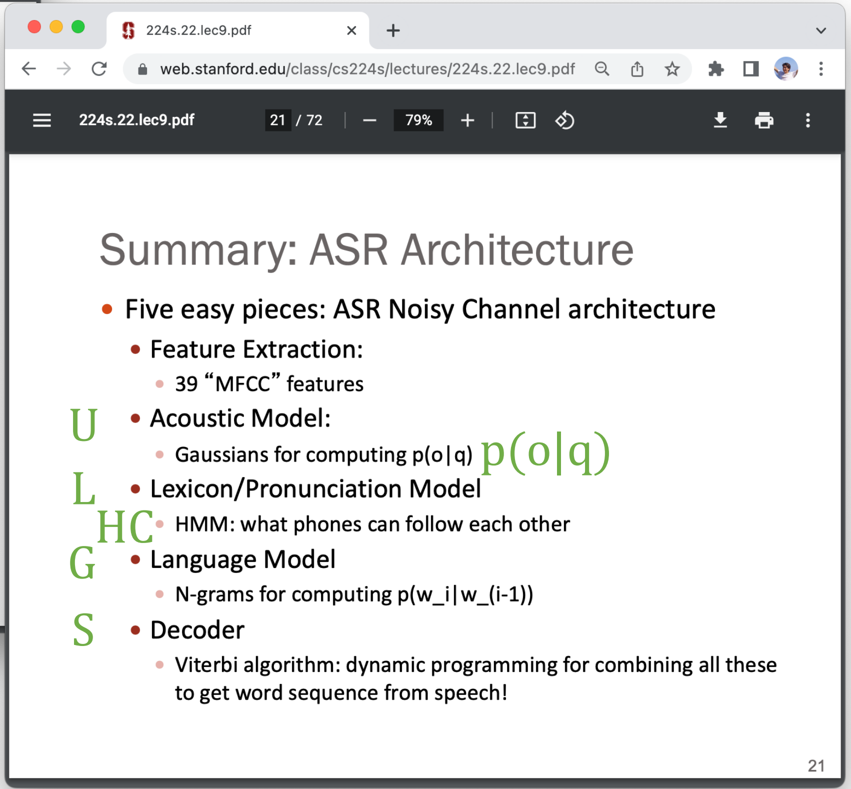
search graph of the utterance



Best path



Composition



WFST ←→ Probability Theory

1. A WFST, R, that translates strings $a \in \mathcal{A}$ into strings $b \in \mathcal{B}$ with ~~joint probability $p(a, b)$~~. p(a|b)
2. Another WFST, S, that translates strings $b \in \mathcal{B}$ into strings $c \in \mathcal{C}$ with ~~conditional probability $p(c|b)$~~. p(b,c)

The operation $T = R \circ S$ gives you a WFST, T, that translates strings $a \in \mathcal{A}$ into strings $c \in \mathcal{C}$ with joint probability

$$p(a, c) = \sum_{b \in \mathcal{B}} \cancel{p(a, b) p(c|b)} \; p(a|b)p(b,c)$$

If S is a WFSA: p(b,b) = p(b)

Noisy Channel ←→ WFST

$$P(O|W)P(W)$$

P(W) G: Grammar (e.g. trigram LM)

$$\sum_{L} P(O|L)P(L|W)P(W)$$

P(L|W) L: Lexicon (pronunciation dictionary)

$$\sum_{C,L} P(O|C)P(C|L)P(L|W)P(W)$$

P(C|L) C: Context-dependency (decision tree)

$$\sum_{Q,C,L} P(O|Q)P(Q|C)P(C|L)P(L|W)P(W)$$

P(Q|C) H: HMM (e.g. biphones)

$$U \circ H \circ C \circ L \circ G$$

U: AM G: LM

$H \circ C \circ L \circ G$: graph

Viterbi Approximation

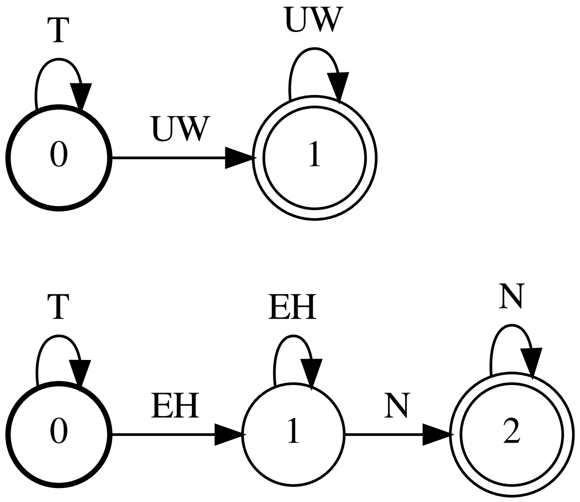
$$\underset{W}{\operatorname{argmax}} \sum_{Q} P(O|Q)P(Q|W)P(W) \cong \underset{Q,W}{\operatorname{argmax}} P(O|Q)P(Q|W)P(W)$$

$$\underset{W}{\operatorname{argmax}} \sum_{Q,C,L} P(O|Q)P(Q|C)P(C|L)P(L|W)P(W) \cong \underset{Q,C,L,W}{\operatorname{argmax}} P(O|Q)P(Q|C)P(C|L)P(L|W)P(W)$$

$$= \mathrm{BestPath}(U \circ H \circ C \circ L \circ G)$$

Example

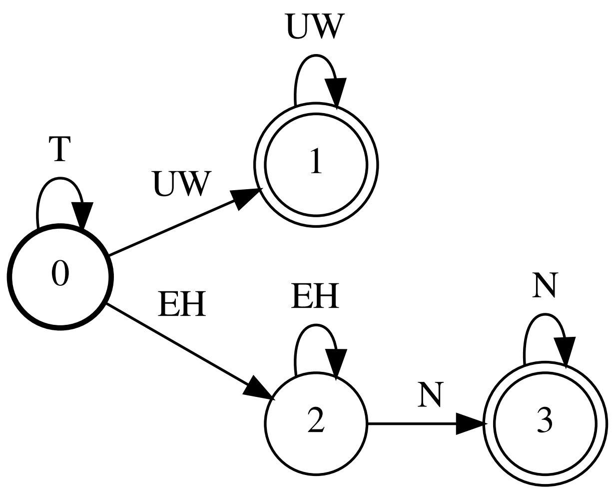
- Build a graph for each word.



Credit: Adam Janin, CS188 (Berkeley)

Example

- Build a graph for each word.
- Combine where possible.



Credit: Adam Janin, CS188 (Berkeley)

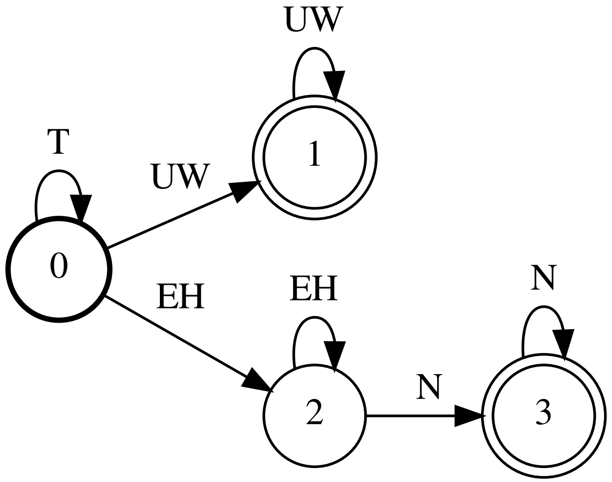
Example

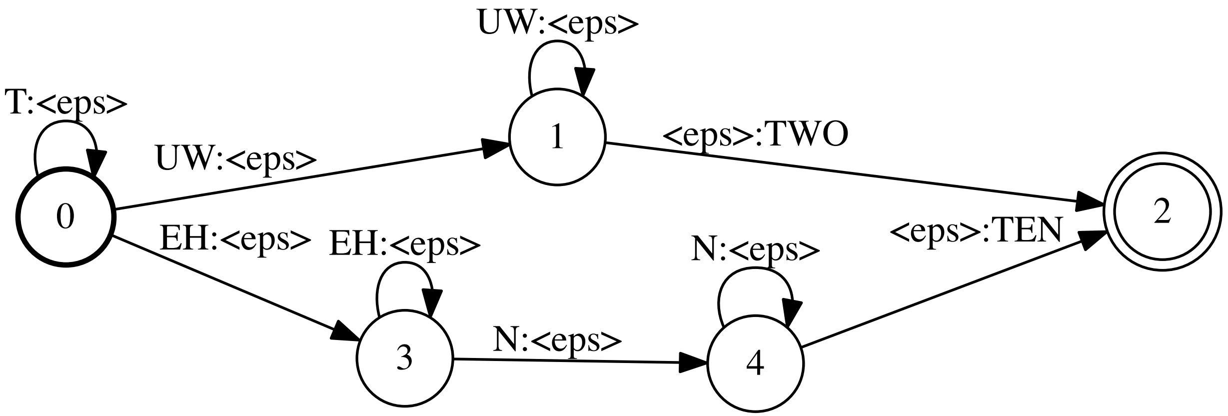
x:y – When you traverse the arc, consume "x" and emit "y".

<eps> - Epsilon.

- On input, do not consume any input.
- On output, do not emit any output.

For any word/pronunciation: all input is consumed, one word is output.



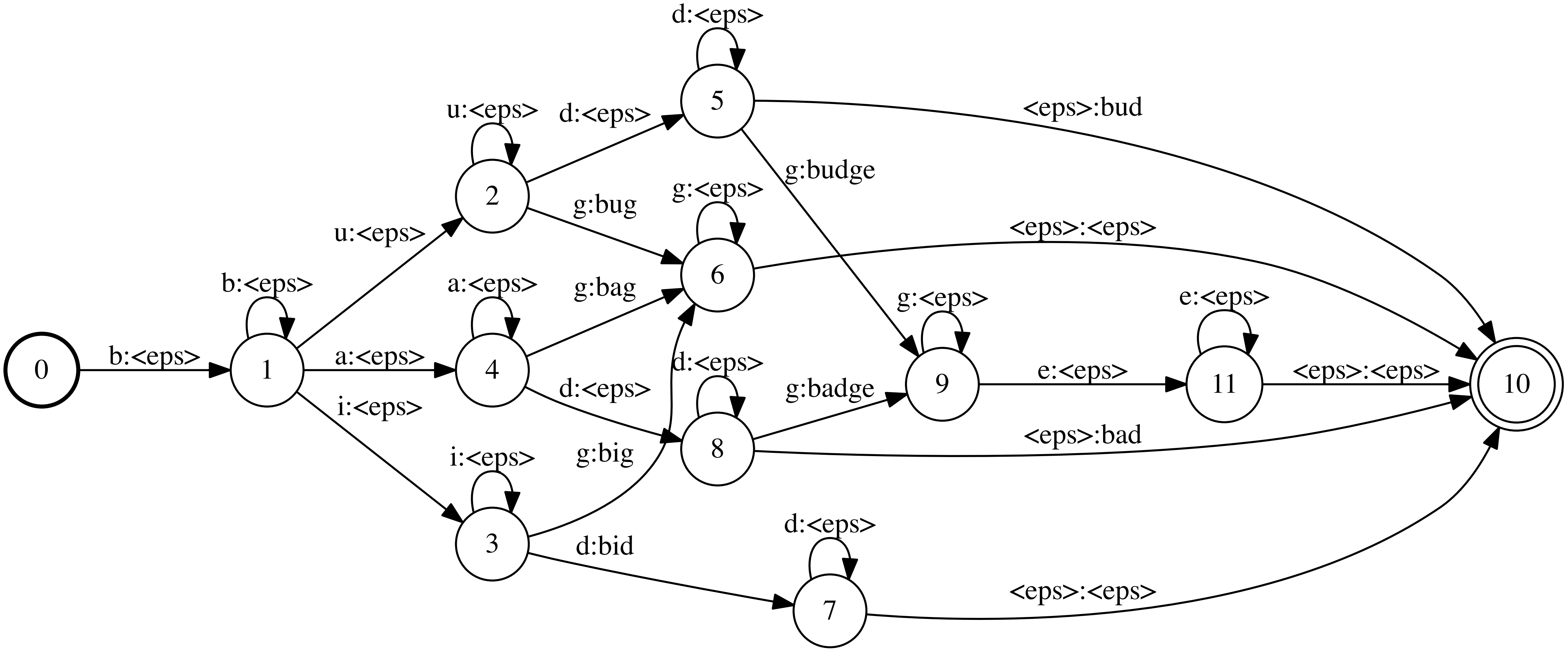


Credit: Adam Janin, CS188 (Berkeley)

Example

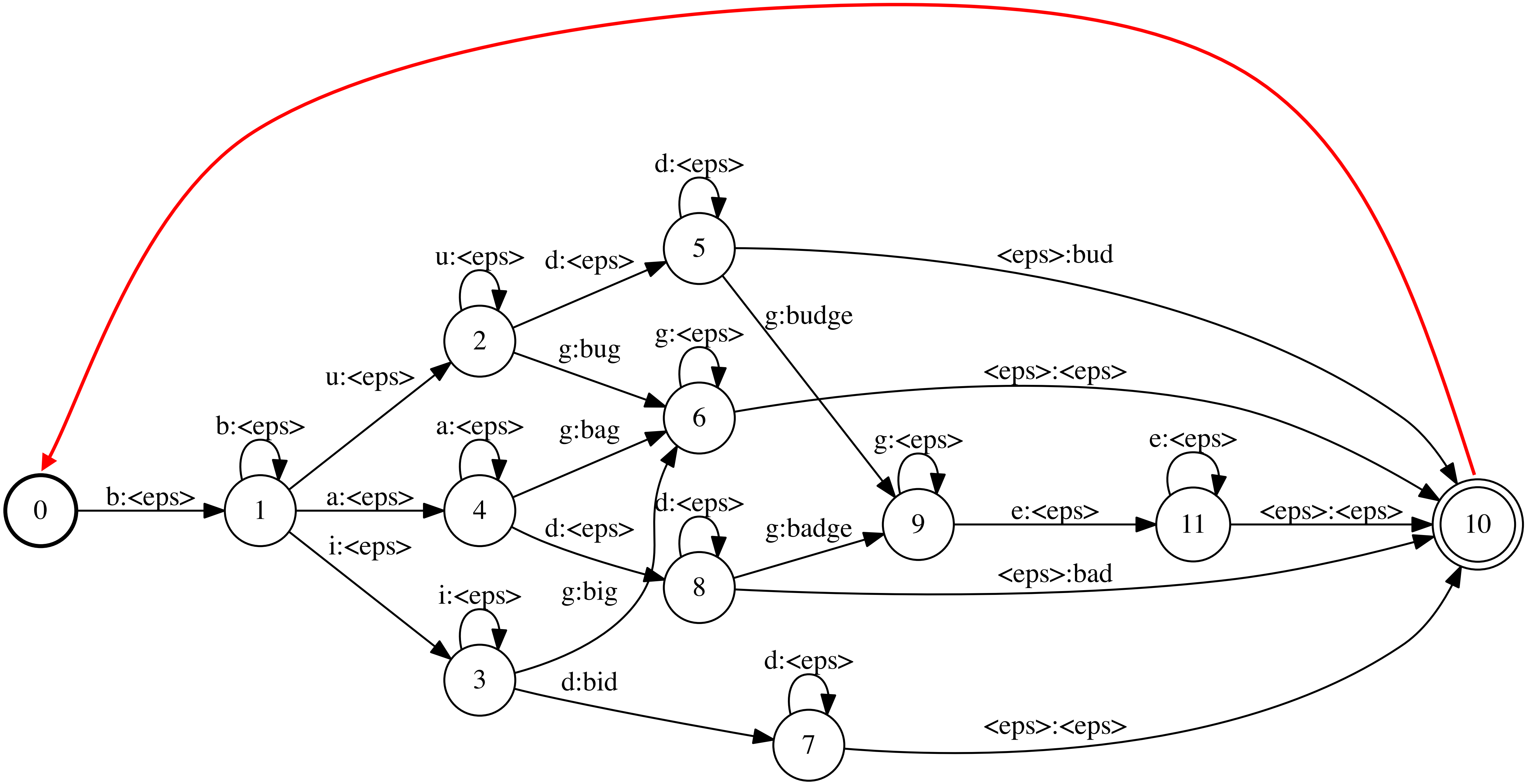
- Next slide has a bigger example:
  - **bad, badge, bag, bid, big, bud, budge, bug**
- Uses letters rather than phonemes to make it easier to read.
- The data structure is known as a “decoding graph”.

Credit: Adam Janin, CS188 (Berkeley)



bad, badge, bag, bid, big, bud, budge, bug

Credit: Adam Janin, CS188 (Berkeley)



Credit: Adam Janin, CS188 (Berkeley)

U = DNN output (WFSA)

(an acceptor is represented as a WFST with identical input and output symbols). It has $T+1$ states, with an arc for each combination of (time, context-dependent HMM state). The costs on these arcs correspond to negated and scaled acoustic log-likelihoods.

p(o|q) = U

ShortestPath = BestPath

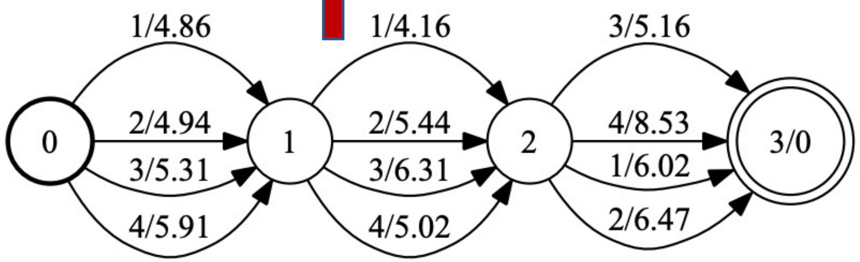


Fig. 1. Acceptor U describing the acoustic scores of an utterance

HCLG = decoding graph (WFST)

$$HCLG = \min(\det(H \circ C \circ L \circ G))$$

where H, C, L and G represent the HMM structure, phonetic context-dependency, lexicon and grammar respectively, and $\circ$ is WFST composition (note: view $HCLG$ as a single symbol).

In $HCLG$, the input labels are the identifiers of context-dependent HMM states, and the output labels represent words.

S = search graph (WFST)

$$S \equiv U \circ HCLG$$

which we call the *search graph* of the utterance. It has approximately $T+1$ times more states than $HCLG$ itself. The decoding problem is equivalent to finding the best path through S. The input symbol sequence for this best path represents the state-level alignment, and the output symbol sequence is the corresponding sentence.

Graph construction: practical concerns

Implementation

Kaldi: disambiguation symbols, word-position-dependent phones, self-loops, ε-removal.
OpenFST: const vs. vector; prune vs. compress; packaged symbol tables; version skew.

Size

large-vocabulary:	~1	GB	HCLG	(static optimization)
large-vocabulary:	~100	MB	HCL∘G	(dynamic lookahead)
custom grammar:	~1	KB	HCLG	(dynamic composition)

Speed

large-vocabulary:	~10	minutes	(single-threaded, ~10G memory)
large-vocabulary:	~0.1	seconds	(add words w/ unigram probability)
custom grammar:	~10	milliseconds	(may even be network bound)

Graph search: practical concerns

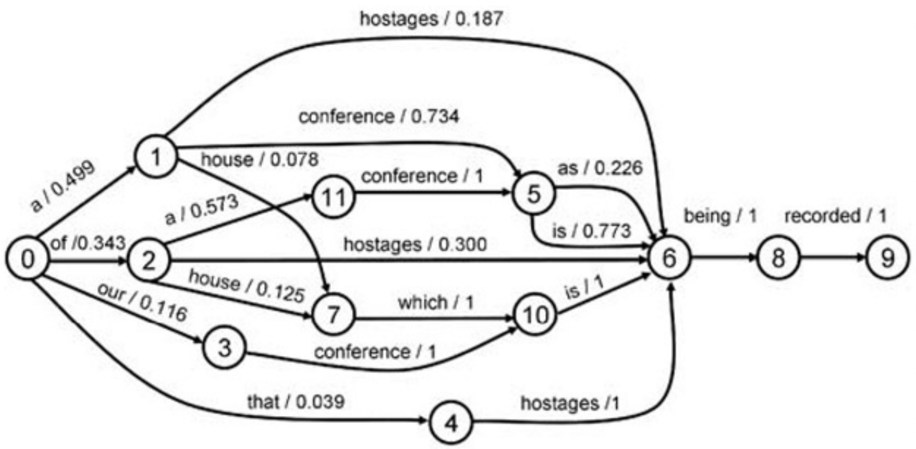
we do not do a full search of S, but use beam pruning. Let B be the searched subset of S, containing a subset of the states and arcs of S obtained by some heuristic pruning procedure. When we do Viterbi decoding with beam-pruning, we are finding the best path through B. Since the beam pruning is a part of any practical search procedure and cannot easily be avoided, we will define the desired outcome of lattice generation in terms of the visited subset B of S.

Settings: pruning beam, lattice beam, max active states
Profiling: not much compute or memory usage (per thread)
Speed: mostly DNN evaluation (matrix multiplication)
Memory: mostly lattice determinization (if needed)

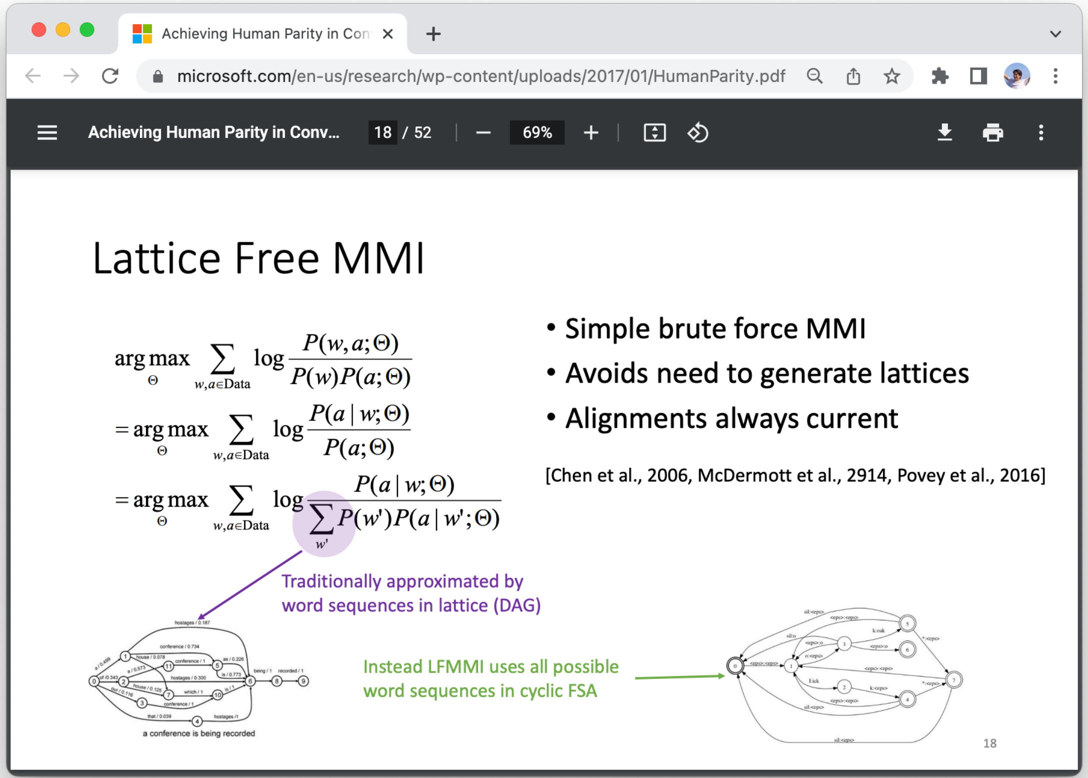
Lattice definition

tl;dr: directed acyclic weighted word graph ("DAWWG")

There is no generally accepted single definition of a lattice. In [3] and [4], it is defined as a labeled, weighted, directed acyclic graph (i.e. a WFSA, with word labels).



a conference is being recorded



Text: danielpovey.com/files/2012_icassp_lattices.pdf
Figure: (source unknown; perhaps Murat Saraçlar, AT&T, 2004)

Lattice properties

tl;dr: $\hat{W} \in L \subseteq B \subset S$

- The lattice should have a path for every word sequence within α of the best-scoring one.
- The scores and alignments in the lattice should be accurate.
- The lattice should not contain duplicate paths with the same word sequence.

Text: danielpovey.com/files/2012_icassp_lattices.pdf

Lattice generation

5.4. Summary of our algorithm

During decoding, we create a data-structure corresponding to a full state-level lattice. That is, for every arc of $HCLG$, we traverse on every frame, we create a separate arc in the state-level lattice. These arcs contain the acoustic and graph costs separately. We prune the state-level graph using a beam α; we do this periodically (every 25 frames) but this is equivalent to doing it just once at the end, as in [3]. Let the final pruned state-level lattice be P. Let $Q = \mathrm{inv}(P)$, and let E be an encoded version of Q as described above (with the state labels as part of the weights). The final lattice is

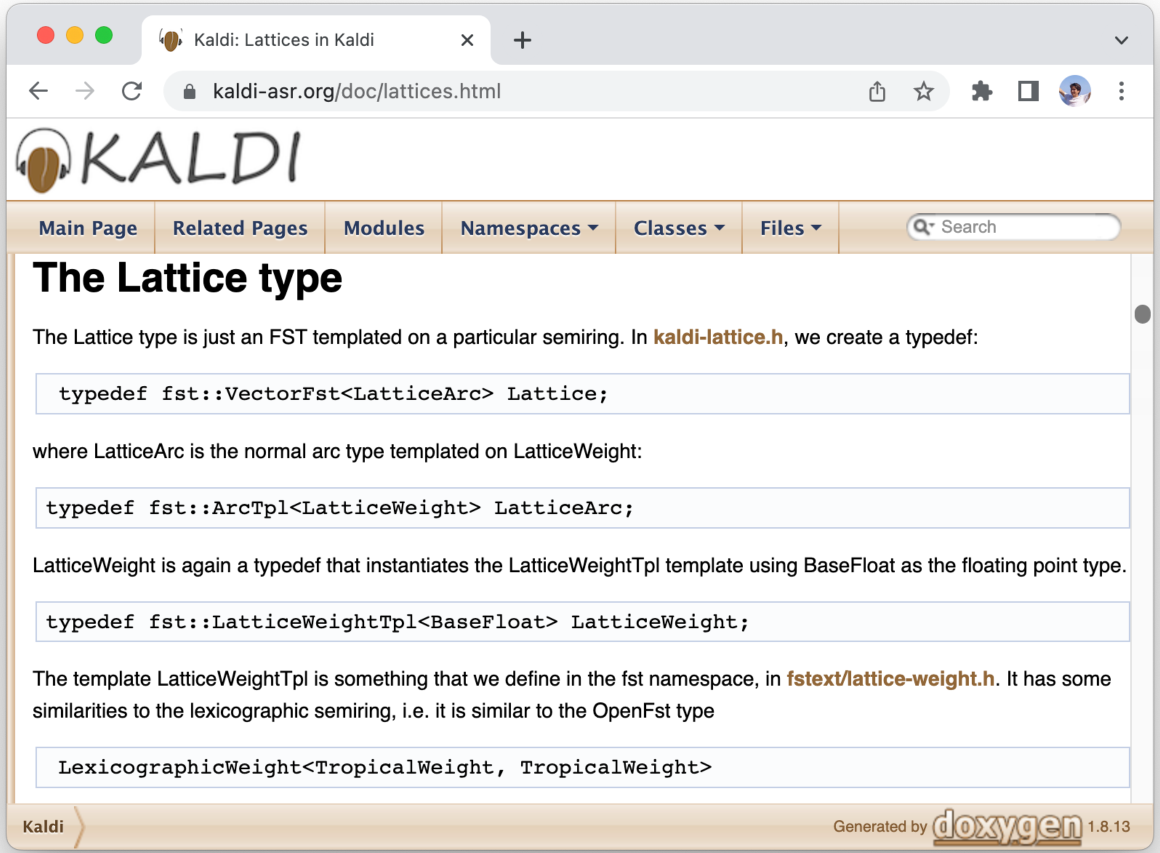
$$L = \mathrm{prune}(\det(\mathrm{rmeps}(E)), \alpha). \tag{4}$$

The determinization and epsilon removal are done together by a single algorithm that we will describe below. L is a deterministic, acyclic weighted acceptor with the words as the labels, and the graph and acoustic costs and the alignments encoded into the weights. The costs and alignments are not "synchronized" with the words.

Text: danielpovey.com/files/2012_icassp_lattices.pdf

Lattices in Kaldi

tl;dr: `LatticeWeight=(graph_cost,am_cost)`



Kaldi: Lattices in Kaldi

kaldi-asr.org/doc/lattices.html

KALDI

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Search

The Lattice type

The Lattice type is just an FST templated on a particular semiring. In kaldi-lattice.h, we create a typedef:

```
typedef fst::VectorFst<LatticeArc> Lattice;
```

where LatticeArc is the normal arc type templated on LatticeWeight:

```
typedef fst::ArcTpl<LatticeWeight> LatticeArc;
```

LatticeWeight is again a typedef that instantiates the LatticeWeightTpl template using BaseFloat as the floating point type.

```
typedef fst::LatticeWeightTpl<BaseFloat> LatticeWeight;
```

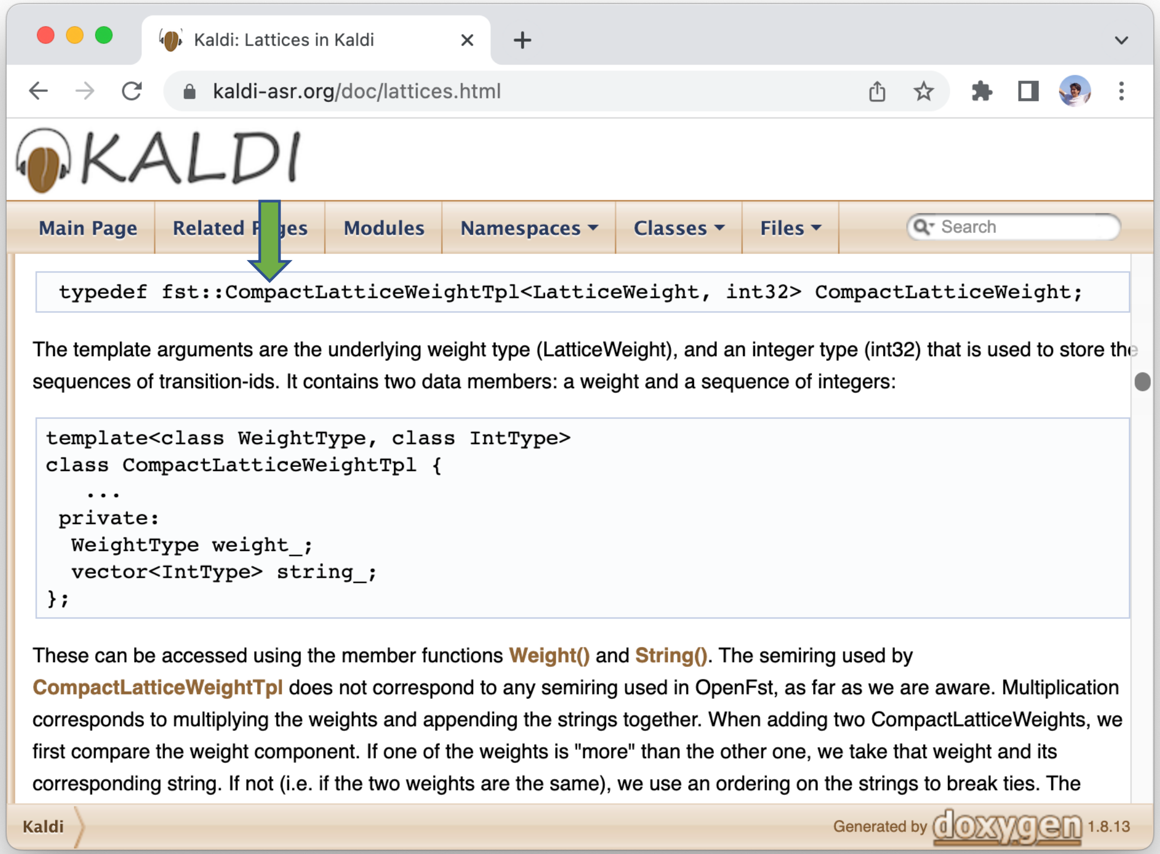
The template LatticeWeightTpl is something that we define in the fst namespace, in fstext/lattice-weight.h. It has some similarities to the lexicographic semiring, i.e. it is similar to the OpenFst type

```
LexicographicWeight<TropicalWeight, TropicalWeight>
```

Kaldi

Generated by doxygen 1.8.13

WFST



Kaldi: Lattices in Kaldi

kaldi-asr.org/doc/lattices.html

KALDI

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Search

```
typedef fst::CompactLatticeWeightTpl<LatticeWeight, int32> CompactLatticeWeight;
```

The template arguments are the underlying weight type (LatticeWeight), and an integer type (int32) that is used to store the sequences of transition-ids. It contains two data members: a weight and a sequence of integers:

```
template<class WeightType, class IntType>
class CompactLatticeWeightTpl {
   ...
 private:
  WeightType weight_;
  vector<IntType> string_;
};
```

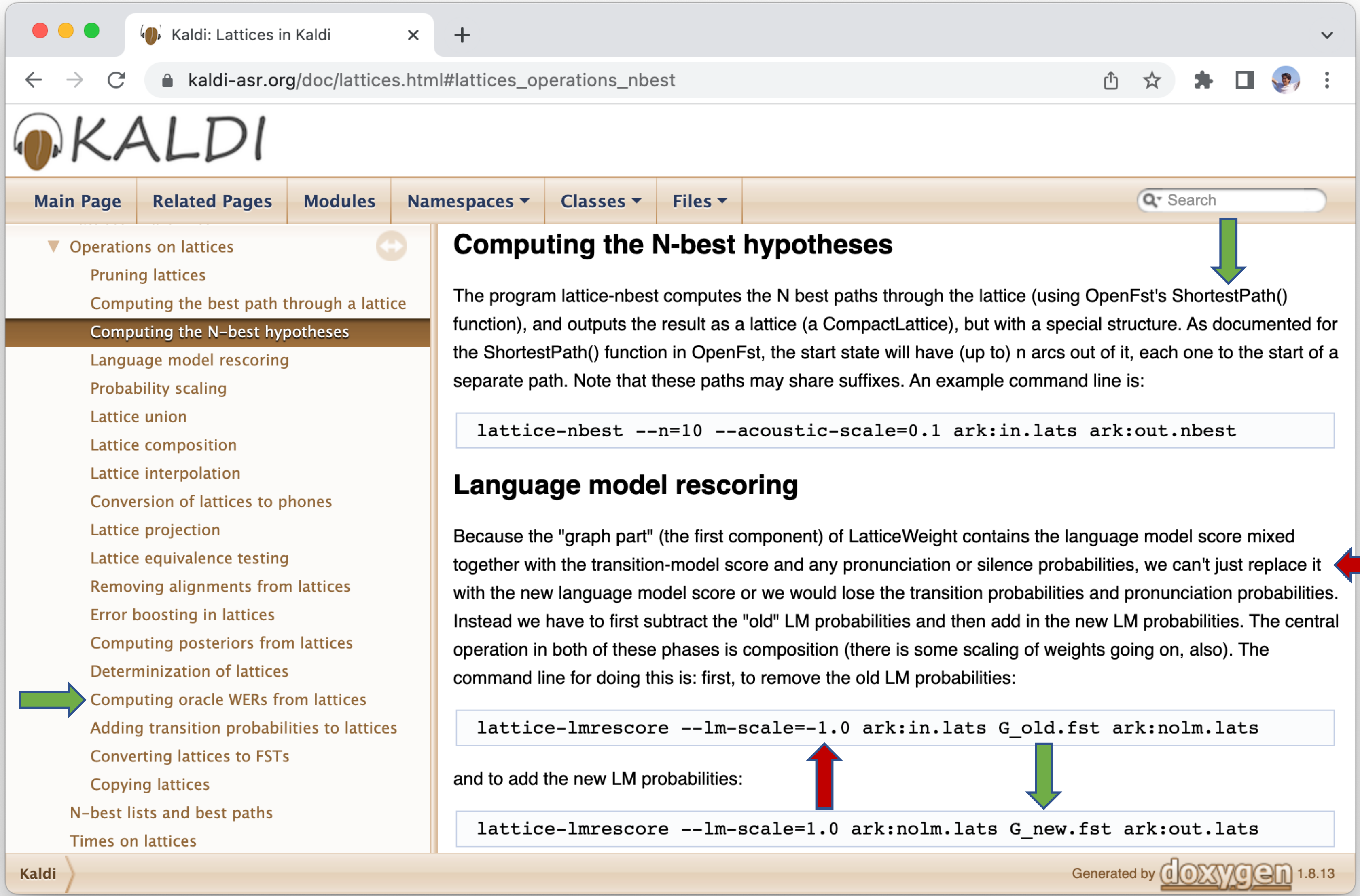
These can be accessed using the member functions Weight() and String(). The semiring used by CompactLatticeWeightTpl does not correspond to any semiring used in OpenFst, as far as we are aware. Multiplication corresponds to multiplying the weights and appending the strings together. When adding two CompactLatticeWeights, we first compare the weight component. If one of the weights is "more" than the other one, we take that weight and its corresponding string. If not (i.e. if the two weights are the same), we use an ordering on the strings to break ties. The

Kaldi

Generated by doxygen 1.8.13

WFSA

Lattice operations



Kaldi: Lattices in Kaldi

kaldi-asr.org/doc/lattices.html#lattices_operations_nbest

KALDI

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Search

Computing the N-best hypotheses

The program lattice-nbest computes the N best paths through the lattice (using OpenFst's ShortestPath() function), and outputs the result as a lattice (a CompactLattice), but with a special structure. As documented for the ShortestPath() function in OpenFst, the start state will have (up to) n arcs out of it, each one to the start of a separate path. Note that these paths may share suffixes. An example command line is:

```
lattice-nbest --n=10 --acoustic-scale=0.1 ark:in.lats ark:out.nbest
```

Language model rescoring

Because the "graph part" (the first component) of LatticeWeight contains the language model score mixed together with the transition-model score and any pronunciation or silence probabilities, we can't just replace it with the new language model score or we would lose the transition probabilities and pronunciation probabilities. Instead we have to first subtract the "old" LM probabilities and then add in the new LM probabilities. The central operation in both of these phases is composition (there is some scaling of weights going on, also). The command line for doing this is: first, to remove the old LM probabilities:

```
lattice-lmrescore --lm-scale=-1.0 ark:in.lats G_old.fst ark:nolm.lats
```

and to add the new LM probabilities:

```
lattice-lmrescore --lm-scale=1.0 ark:nolm.lats G_new.fst ark:out.lats
```

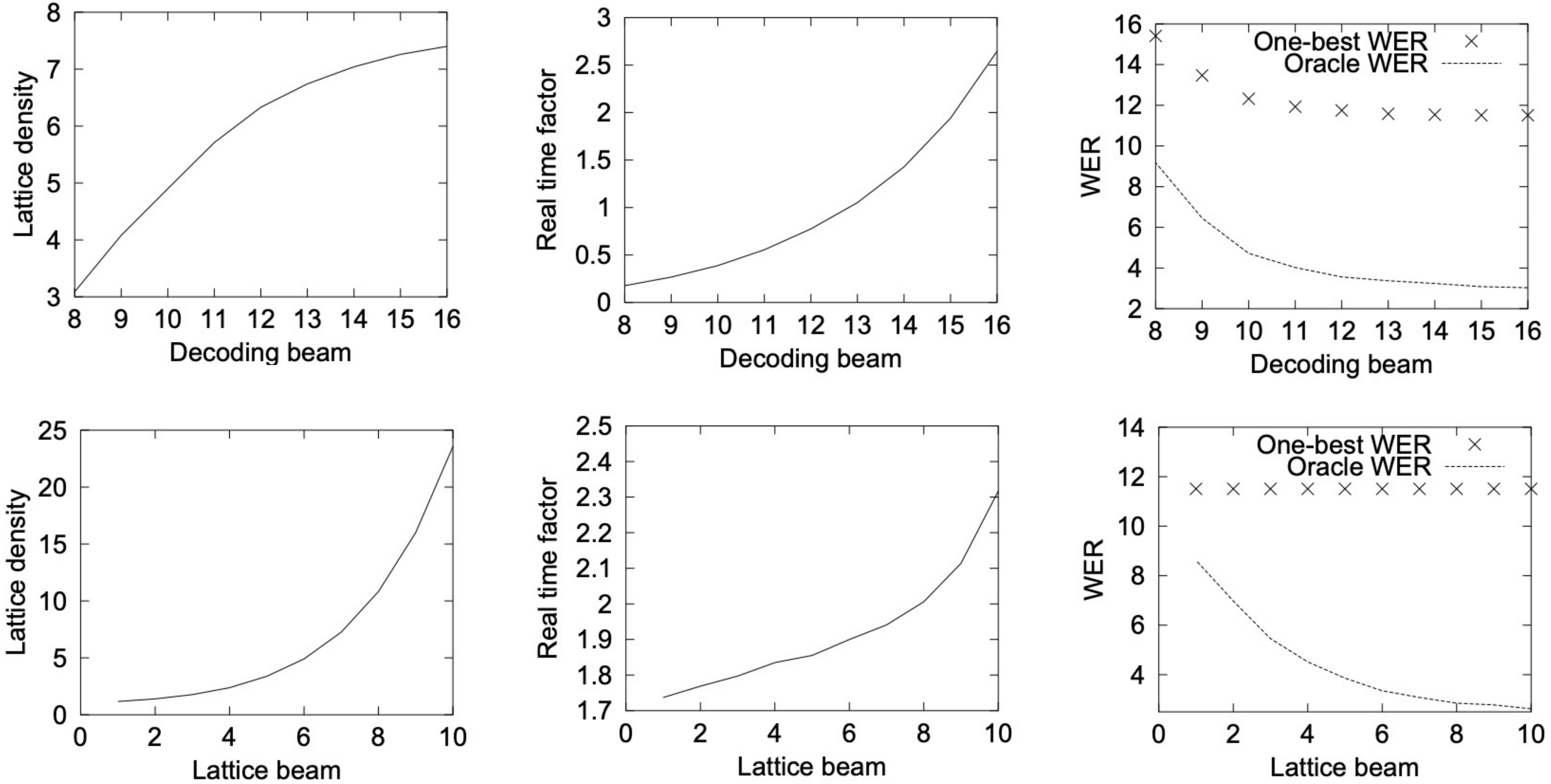
Kaldi

Generated by doxygen 1.8.13

Performance tradeoffs and oracle WER

Lattice beam = 7

Decoding beam = 15



Conclusion

WFST framework enables practical ASR:

1. interpretable sub-models (not E2E)
2. **composition** → customizable graph
3. **graph search** → **lattice** representation

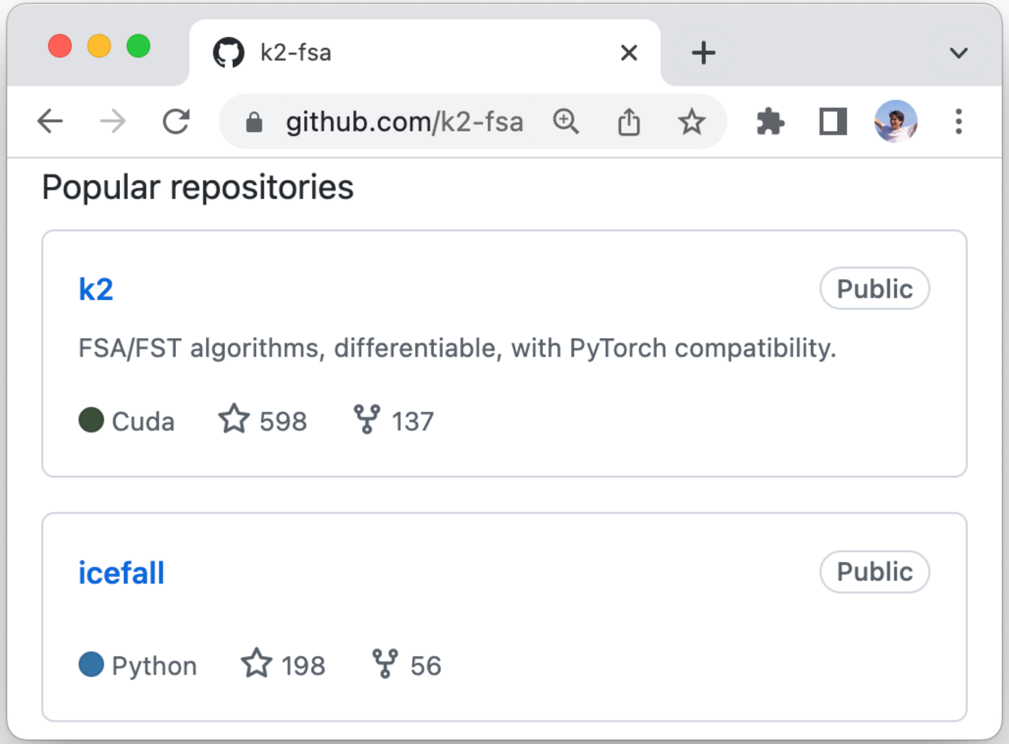
Bonus topics:

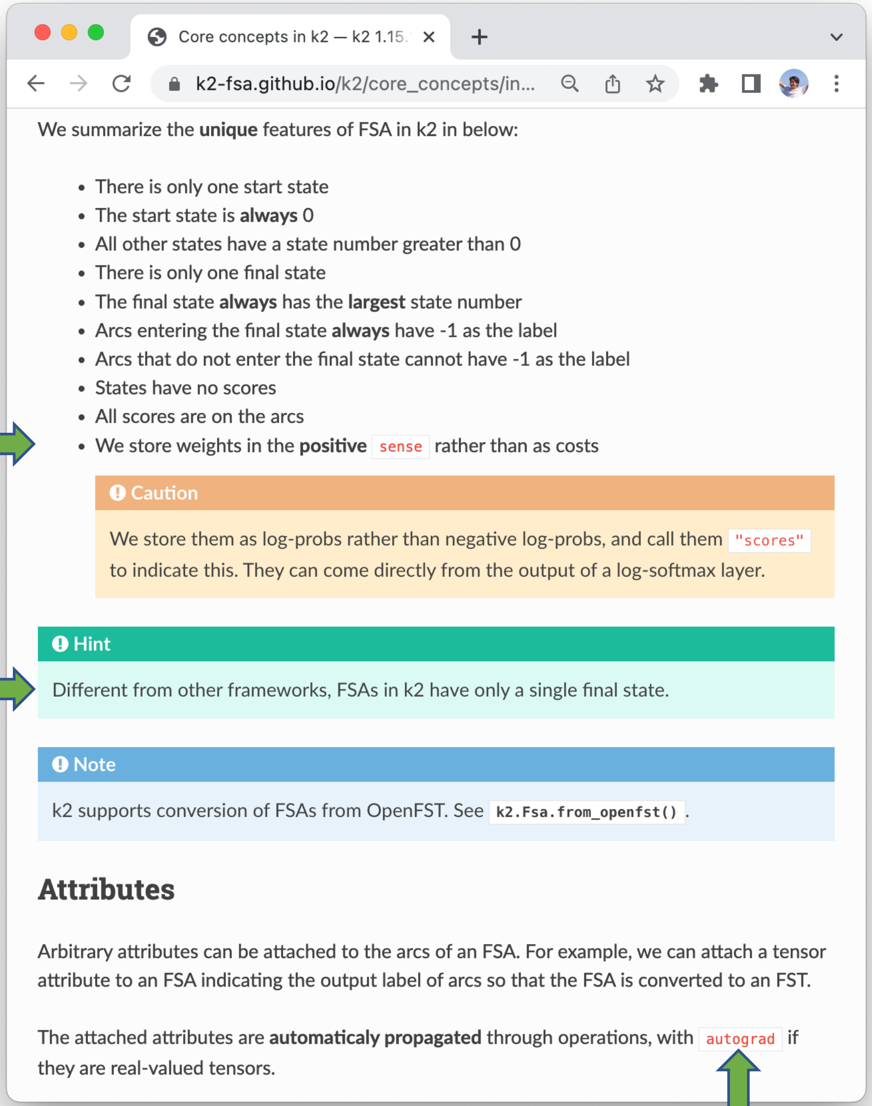
Research: differentiable automata

Demonstration: Mod9 ASR Engine

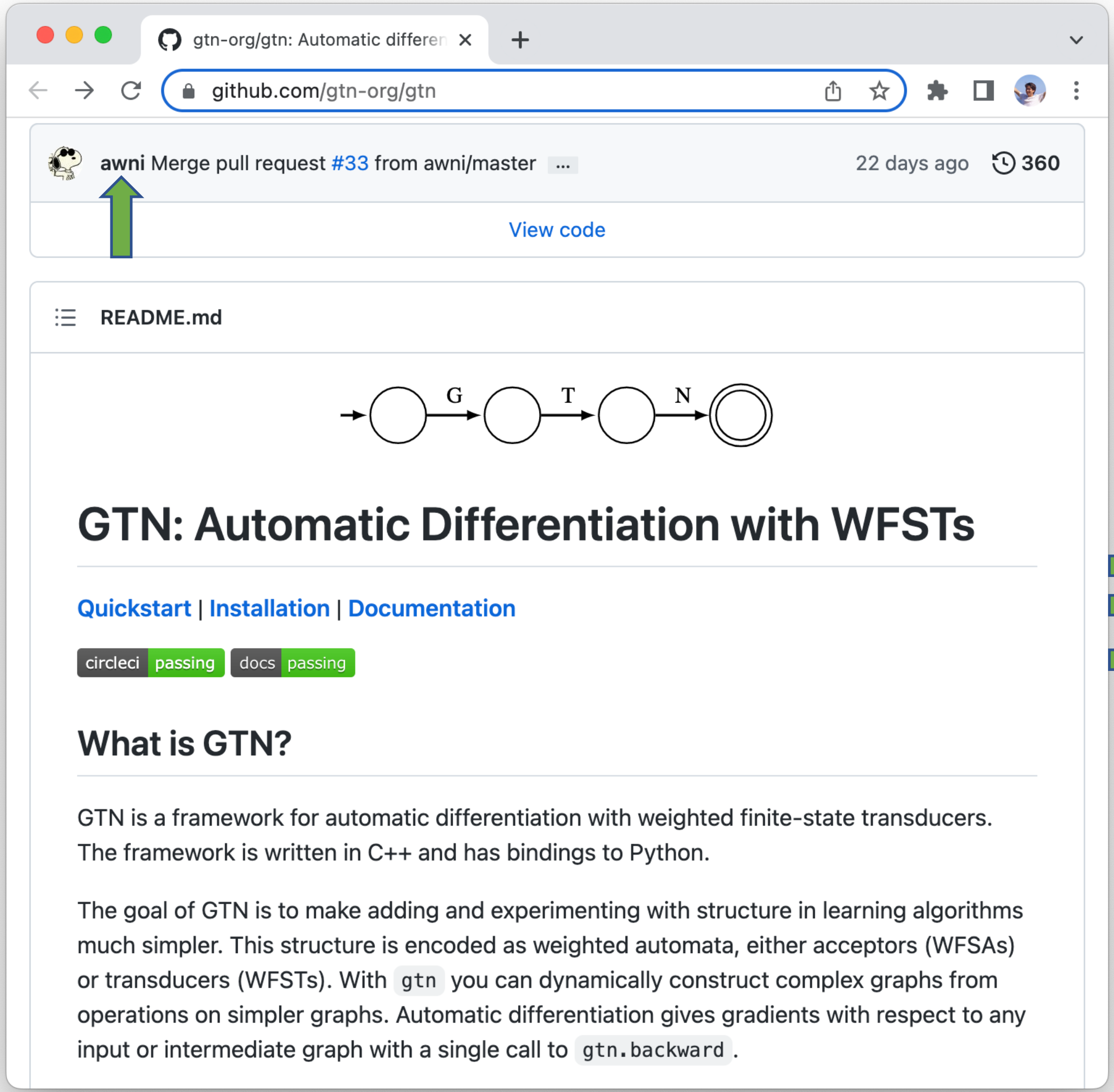
Q&A: e.g. school → startup?

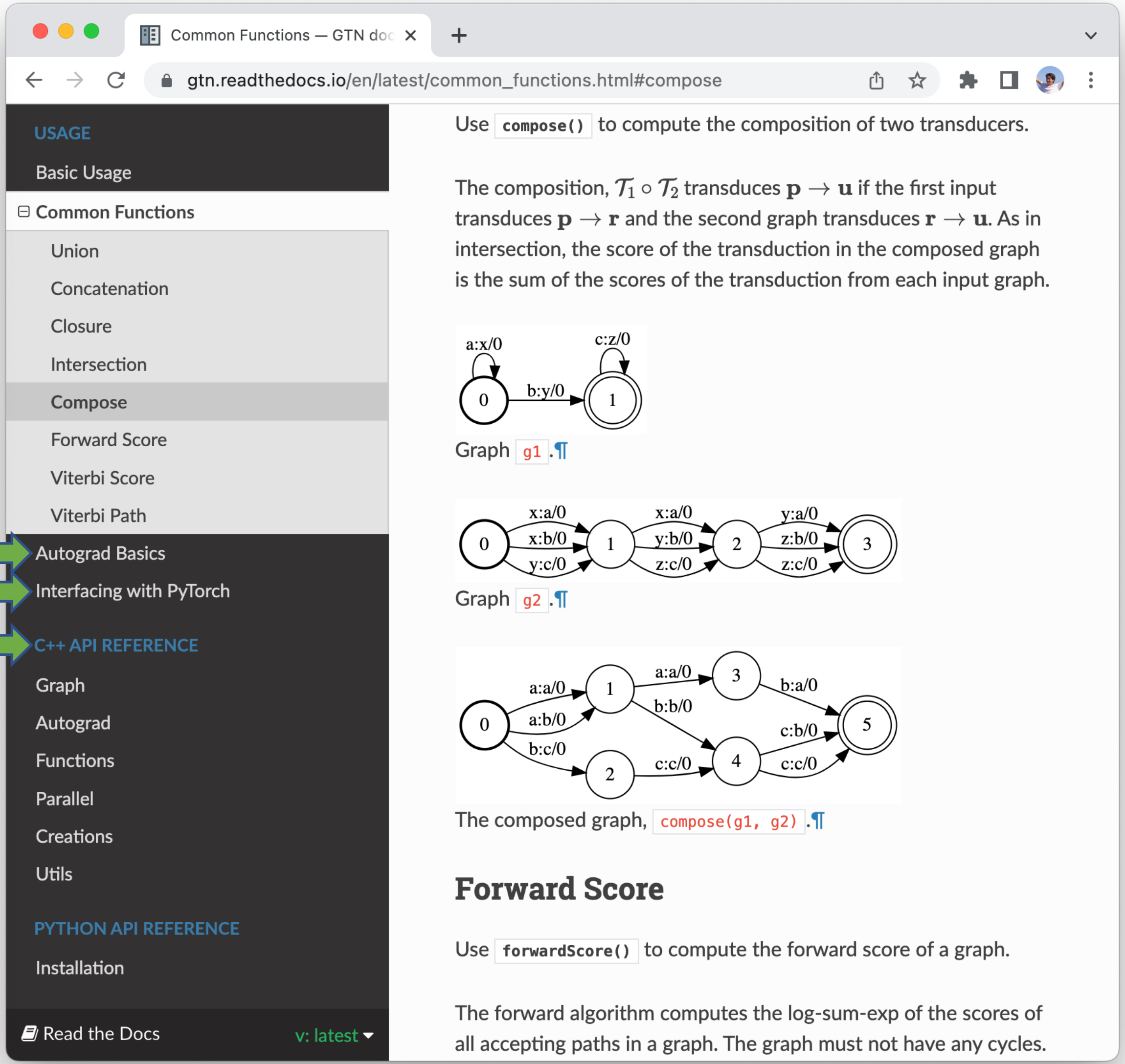
Research: k2-fsa





Research: GTN





Demos

1. **Negative latency for real-time streaming**
   Due to determinization during graph construction
   Also affected by the DNN acoustic model's right context

2. **Switchboard Benchmark**
   Kaldi's egs/fisher_swbd: competitive with cloud platforms
   Lattice representations: oracle performance <1% WER

3. **Dynamic customization**
   Pre-decoding: Add new words to the graph
   Post-decoding: Bias phrases in the lattice

Toward Zero Oracle WER on Switchboard

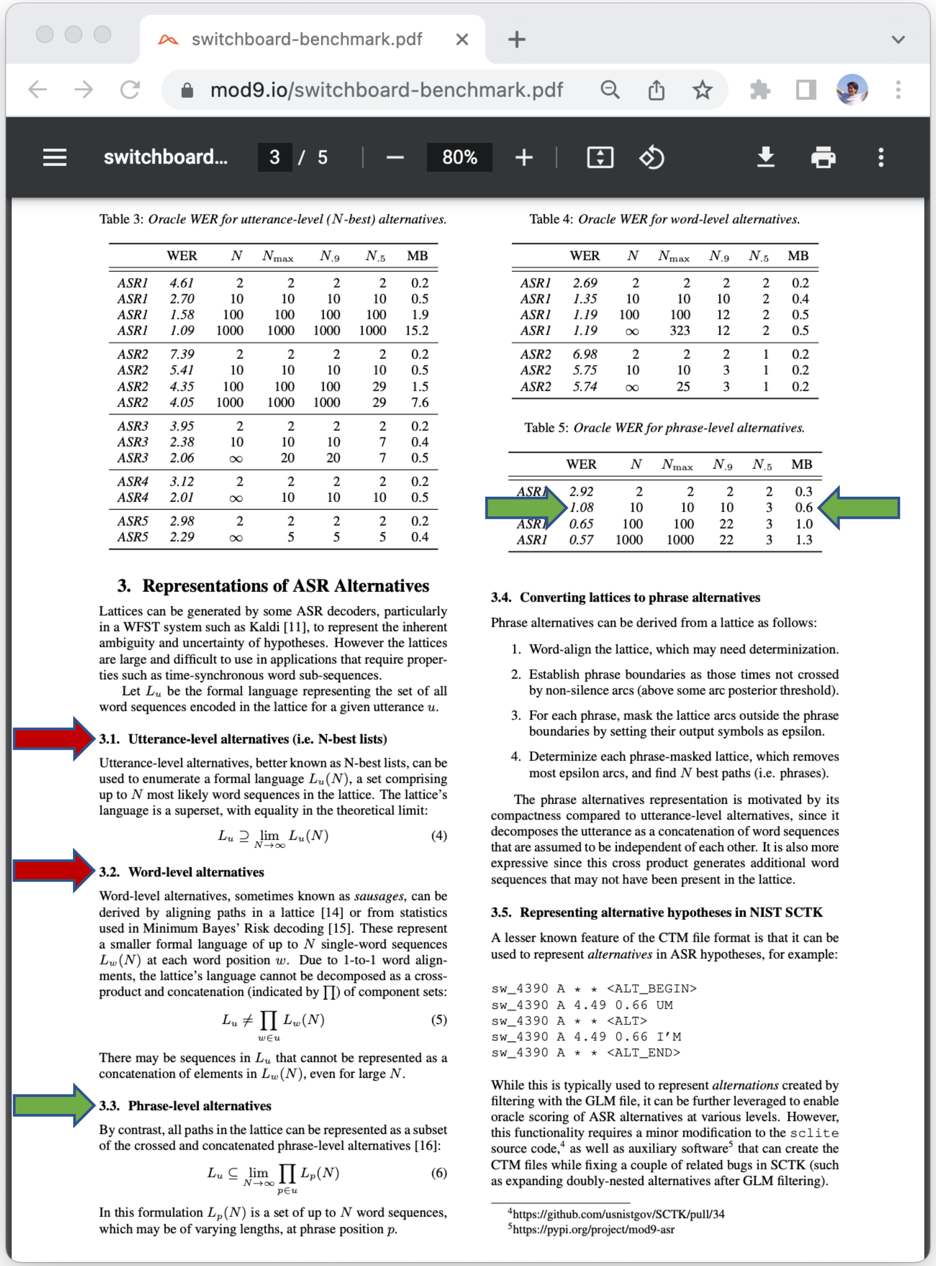


Table 3: *Oracle WER for utterance-level (N-best) alternatives.*

	WER	N	$N_{\max}$	$N_{.9}$	$N_{.5}$	MB
ASR1	*4.61*	2	2	2	2	0.2
ASR1	*2.70*	10	10	10	10	0.5
ASR1	*1.58*	100	100	100	100	1.9
ASR1	*1.09*	1000	1000	1000	1000	15.2
ASR2	*7.39*	2	2	2	2	0.2
ASR2	*5.41*	10	10	10	10	0.5
ASR2	*4.35*	100	100	100	29	1.5
ASR2	*4.05*	1000	1000	1000	29	7.6
ASR3	*3.95*	2	2	2	2	0.2
ASR3	*2.38*	10	10	10	7	0.4
ASR3	*2.06*	∞	20	20	7	0.5
ASR4	*3.12*	2	2	2	2	0.2
ASR4	*2.01*	∞	10	10	10	0.5
ASR5	2.98	2	2	2	2	0.2
ASR5	2.29	∞	5	5	5	0.4

Table 4: *Oracle WER for word-level alternatives.*

	WER	N	$N_{\max}$	$N_{.9}$	$N_{.5}$	MB
ASR1	*2.69*	2	2	2	2	0.2
ASR1	*1.35*	10	10	10	2	0.4
ASR1	*1.19*	100	100	12	2	0.5
ASR1	*1.19*	∞	323	12	2	0.5
ASR2	*6.98*	2	2	2	1	0.2
ASR2	*5.75*	10	10	3	1	0.2
ASR2	*5.74*	∞	25	3	1	0.2

Table 5: *Oracle WER for phrase-level alternatives.*

	WER	N	$N_{\max}$	$N_{.9}$	$N_{.5}$	MB
ASR1	*2.92*	2	2	2	2	0.3
[illegible]	*1.08*	10	10	10	3	0.6
ASR1	*0.65*	100	100	22	3	1.0
ASR1	*0.57*	1000	1000	22	3	1.3

3. Representations of ASR Alternatives

Lattices can be generated by some ASR decoders, particularly in a WFST system such as Kaldi [11], to represent the inherent ambiguity and uncertainty of hypotheses. However the lattices are large and difficult to use in applications that require properties such as time-synchronous word sub-sequences.

Let L_u be the formal language representing the set of all word sequences encoded in the lattice for a given utterance u.

3.1. Utterance-level alternatives (i.e. N-best lists)

Utterance-level alternatives, better known as N-best lists, can be used to enumerate a formal language $L_u(N)$, a set comprising up to N most likely word sequences in the lattice. The lattice's language is a superset, with equality in the theoretical limit:

$$L_u \supseteq \lim_{N\to\infty} L_u(N) \tag{4}$$

3.2. Word-level alternatives

Word-level alternatives, sometimes known as *sausages*, can be derived by aligning paths in a lattice [14] or from statistics used in Minimum Bayes' Risk decoding [15]. These represent a smaller formal language of up to N single-word sequences $L_w(N)$ at each word position w. Due to 1-to-1 word alignments, the lattice's language cannot be decomposed as a cross-product and concatenation (indicated by $\prod$) of component sets:

$$L_u \neq \prod_{w\in u} L_w(N) \tag{5}$$

There may be sequences in L_u that cannot be represented as a concatenation of elements in $L_w(N)$, even for large N.

3.3. Phrase-level alternatives

By contrast, all paths in the lattice can be represented as a subset of the crossed and concatenated phrase-level alternatives [16]:

$$L_u \subseteq \lim_{N\to\infty} \prod_{p\in u} L_p(N) \tag{6}$$

In this formulation $L_p(N)$ is a set of up to N word sequences, which may be of varying lengths, at phrase position p.

3.4. Converting lattices to phrase alternatives

Phrase alternatives can be derived from a lattice as follows:

1. Word-align the lattice, which may need determinization.
2. Establish phrase boundaries as those times not crossed by non-silence arcs (above some arc posterior threshold).
3. For each phrase, mask the lattice arcs outside the phrase boundaries by setting their output symbols as epsilon.
4. Determinize each phrase-masked lattice, which removes most epsilon arcs, and find N best paths (i.e. phrases).

The phrase alternatives representation is motivated by its compactness compared to utterance-level alternatives, since it decomposes the utterance as a concatenation of word sequences that are assumed to be independent of each other. It is also more expressive since this cross product generates additional word sequences that may not have been present in the lattice.

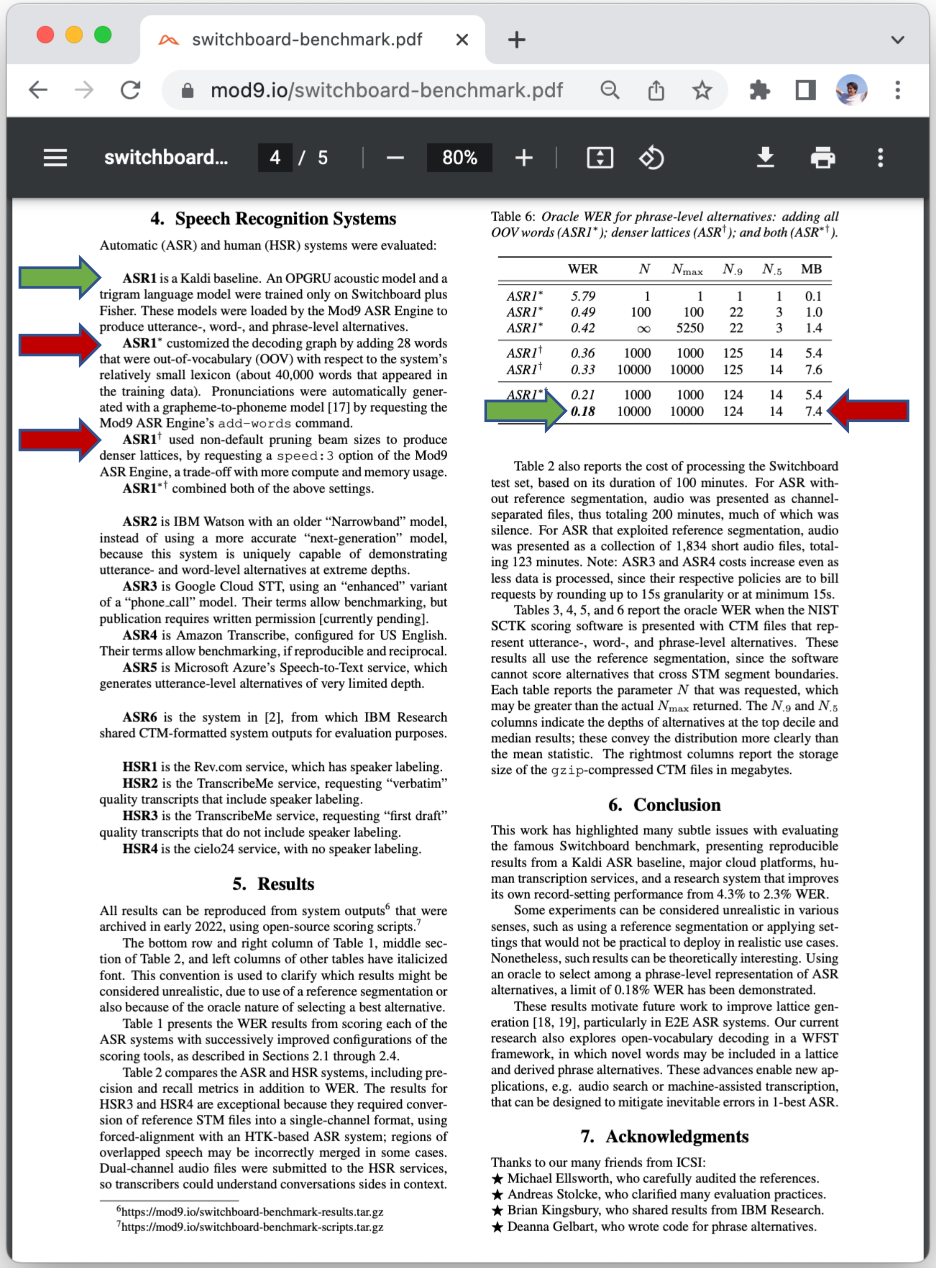
3.5. Representing alternative hypotheses in NIST SCTK

A lesser known feature of the CTM file format is that it can be used to represent *alternatives* in ASR hypotheses, for example:

```
sw_4390 A * * <ALT_BEGIN>
sw_4390 A 4.49 0.66 UM
sw_4390 A * * <ALT>
sw_4390 A 4.49 0.66 I'M
sw_4390 A * * <ALT_END>
```

While this is typically used to represent *alternations* created by filtering with the GLM file, it can be further leveraged to enable oracle scoring of ASR alternatives at various levels. However, this functionality requires a minor modification to the sclite source code,[4] as well as auxiliary software[5] that can create the CTM files while fixing a couple of related bugs in SCTK (such as expanding doubly-nested alternatives after GLM filtering).

[4] https://github.com/usnistgov/SCTK/pull/34
[5] https://pypi.org/project/mod9-asr



4. Speech Recognition Systems

Automatic (ASR) and human (HSR) systems were evaluated:

ASR1 is a Kaldi baseline. An OPGRU acoustic model and a trigram language model were trained only on Switchboard plus Fisher. These models were loaded by the Mod9 ASR Engine to produce utterance-, word-, and phrase-level alternatives.

ASR1* customized the decoding graph by adding 28 words that were out-of-vocabulary (OOV) with respect to the system's relatively small lexicon (about 40,000 words that appeared in the training data). Pronunciations were automatically generated with a grapheme-to-phoneme model [17] by requesting the Mod9 ASR Engine's `add-words` command.

ASR1† used non-default pruning beam sizes to produce denser lattices, by requesting a `speed:3` option of the Mod9 ASR Engine, a trade-off with more compute and memory usage.

ASR1*† combined both of the above settings.

ASR2 is IBM Watson with an older "Narrowband" model, instead of using a more accurate "next-generation" model, because this system is uniquely capable of demonstrating utterance- and word-level alternatives at extreme depths.

ASR3 is Google Cloud STT, using an "enhanced" variant of a "phone_call" model. Their terms allow benchmarking, but publication requires written permission [currently pending].

ASR4 is Amazon Transcribe, configured for US English. Their terms allow benchmarking, if reproducible and reciprocal.

ASR5 is Microsoft Azure's Speech-to-Text service, which generates utterance-level alternatives of very limited depth.

ASR6 is the system in [2], from which IBM Research shared CTM-formatted system outputs for evaluation purposes.

HSR1 is the Rev.com service, which has speaker labeling.

HSR2 is the TranscribeMe service, requesting "verbatim" quality transcripts that include speaker labeling.

HSR3 is the TranscribeMe service, requesting "first draft" quality transcripts that do not include speaker labeling.

HSR4 is the cielo24 service, with no speaker labeling.

5. Results

All results can be reproduced from system outputs[6] that were archived in early 2022, using open-source scoring scripts.[7]

The bottom row and right column of Table 1, middle section of Table 2, and left columns of other tables have italicized font. This convention is used to clarify which results might be considered unrealistic, due to use of a reference segmentation or also because of the oracle nature of selecting a best alternative.

Table 1 presents the WER results from scoring each of the ASR systems with successively improved configurations of the scoring tools, as described in Sections 2.1 through 2.4.

Table 2 compares the ASR and HSR systems, including precision and recall metrics in addition to WER. The results for HSR3 and HSR4 are exceptional because they required conversion of reference STM files into a single-channel format, using forced-alignment with an HTK-based ASR system; regions of overlapped speech may be incorrectly merged in some cases. Dual-channel audio files were submitted to the HSR services, so transcribers could understand conversations sides in context.

[6] https://mod9.io/switchboard-benchmark-results.tar.gz
[7] https://mod9.io/switchboard-benchmark-scripts.tar.gz

Table 6: *Oracle WER for phrase-level alternatives: adding all OOV words (ASR1*); denser lattices (ASR†); and both (ASR*†).*

	WER	N	$N_{\max}$	$N_{.9}$	$N_{.5}$	MB
*ASR1**	*5.79*	1	1	1	1	0.1
*ASR1**	*0.49*	100	100	22	3	1.0
*ASR1**	*0.42*	∞	5250	22	3	1.4
ASR1†	*0.36*	1000	1000	125	14	5.4
ASR1†	*0.33*	10000	10000	125	14	7.6
ASR1†*	*0.21*	1000	1000	124	14	5.4
[illegible]	***0.18***	10000	10000	124	14	7.4

Table 2 also reports the cost of processing the Switchboard test set, based on its duration of 100 minutes. For ASR without reference segmentation, audio was presented as channel-separated files, thus totaling 200 minutes, much of which was silence. For ASR that exploited reference segmentation, audio was presented as a collection of 1,834 short audio files, totaling 123 minutes. Note: ASR3 and ASR4 costs increase even as less data is processed, since their respective policies are to bill requests by rounding up to 15s granularity or at minimum 15s.

Tables 3, 4, 5, and 6 report the oracle WER when the NIST SCTK scoring software is presented with CTM files that represent utterance-, word-, and phrase-level alternatives. These results all use the reference segmentation, since the software cannot score alternatives that cross STM segment boundaries. Each table reports the parameter N that was requested, which may be greater than the actual $N_{\max}$ returned. The $N_{.9}$ and $N_{.5}$ columns indicate the depths of alternatives at the top decile and median results; these convey the distribution more clearly than the mean statistic. The rightmost columns report the storage size of the `gzip`-compressed CTM files in megabytes.

6. Conclusion

This work has highlighted many subtle issues with evaluating the famous Switchboard benchmark, presenting reproducible results from a Kaldi ASR baseline, major cloud platforms, human transcription services, and a research system that improves its own record-setting performance from 4.3% to 2.3% WER.

Some experiments can be considered unrealistic in various senses, such as using a reference segmentation or applying settings that would not be practical to deploy in realistic use cases. Nonetheless, such results can be theoretically interesting. Using an oracle to select among a phrase-level representation of ASR alternatives, a limit of 0.18% WER has been demonstrated.

These results motivate future work to improve lattice generation [18, 19], particularly in E2E ASR systems. Our current research also explores open-vocabulary decoding in a WFST framework, in which novel words may be included in a lattice and derived phrase alternatives. These advances enable new applications, e.g. audio search or machine-assisted transcription, that can be designed to mitigate inevitable errors in 1-best ASR.

7. Acknowledgments

Thanks to our many friends from ICSI:
★ Michael Ellsworth, who carefully audited the references.
★ Andreas Stolcke, who clarified many evaluation practices.
★ Brian Kingsbury, who shared results from IBM Research.
★ Deanna Gelbart, who wrote code for phrase alternatives.

Thanks!

MOD9

 v. 1.9.2

```
docker run mod9/asr engine --help
```

help@mod9.io
+1(HUH)ASK-ARLO