Lecture 8

LVCSR Decoding

Bhuvana Ramabhadran, Michael Picheny, Stanley F. Chen

IBM T.J. Watson Research Center
Yorktown Heights, New York, USA
{bhuvana,picheny,stanchen}@us.ibm.com

27 October 2009

Administrivia

- Main feedback from last lecture.
  - Mud: *k*-means clustering.
- Lab 2 handed back today.
  - Answers:
    `/user1/faculty/stanchen/e6870/lab2_ans/`.
- Lab 3 due Thursday, 11:59pm.
- Next week: Election Day.
- Lab 4 out by then?

The Big Picture

- Weeks 1–4: Small vocabulary ASR.
- Weeks 5–8: Large vocabulary ASR.
  - Week 5: Language modeling.
  - Week 6: Pronunciation modeling ⇔ acoustic modeling for large vocabularies.
  - Week 7: Training for large vocabularies.
  - Week 8: Decoding for large vocabularies.
- Weeks 9–13: Advanced topics.

Outline

- Part I: Introduction to LVCSR decoding, *i.e.*, *search*.
- Part II: Finite-state transducers.
- Part III: Making decoding efficient.
- Part IV: Other decoding paradigms.

Part I

Introduction to LVCSR Decoding

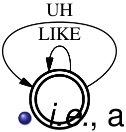
Decoding for LVCSR

$$\begin{aligned}\text{class}(\mathbf{x}) &= \arg\max_{\omega} P(\omega|\mathbf{x}) \\ &= \arg\max_{\omega} \frac{P(\omega)P(\mathbf{x}|\omega)}{P(\mathbf{x})} \\ &= \arg\max_{\omega} P(\omega)P(\mathbf{x}|\omega)\end{aligned}$$

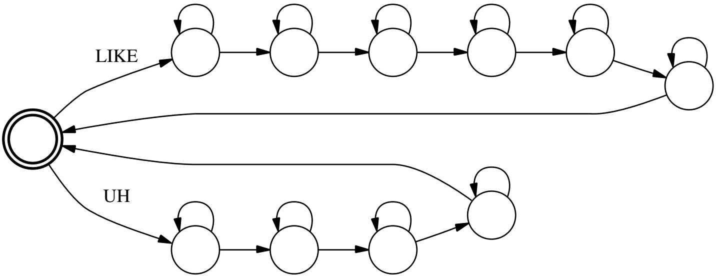
- Now that we know how to build models for LVCSR …
  - *n*-gram models via counting and smoothing.
  - CD acoustic models via complex recipes.
- How can we use them for decoding?

Decoding: Small Vocabulary

- Take graph/WFSA representing language model.



  - *i.e.*, all allowable word sequences.
- Expand to underlying HMM.

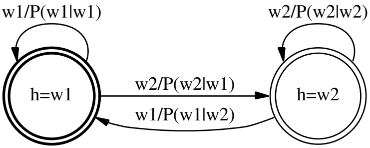


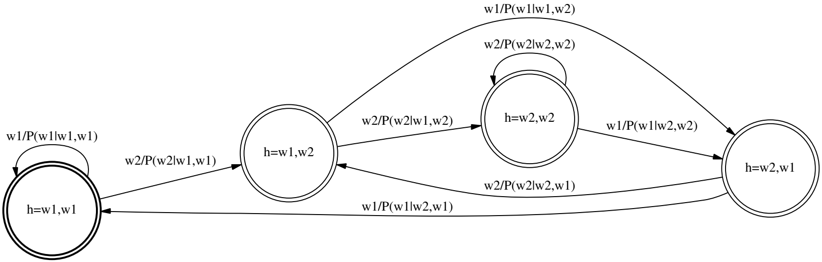
- Run the Viterbi algorithm!

Issue: Are N-Gram Models WFSA's?

- Yup.
- One state for each $(n-1)$-gram history ω.
- All paths ending in state ω ...
  - Are labeled with word sequence ending in ω.
- State ω has outgoing arc for each word w ...
  - With arc probability $P(w|\omega)$.

Bigram, Trigram LM's Over Two Word Vocab



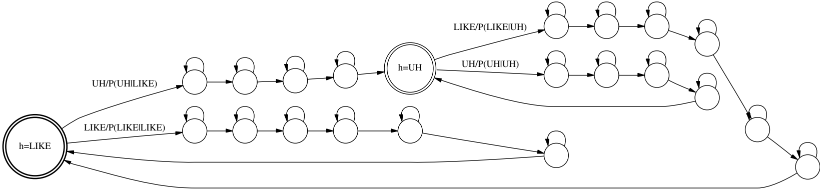


Pop Quiz

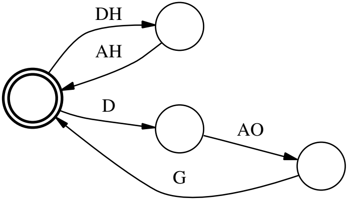
- How many states in FSA representing *n*-gram model ...
  - With vocabulary size $|V|$?
- How many arcs?

Issue: Graph Expansion

- Word models.
  - Replace each word with its HMM.
- CI phone models.
  - Replace each word with its phone sequence(s).
  - Replace each phone with its HMM.

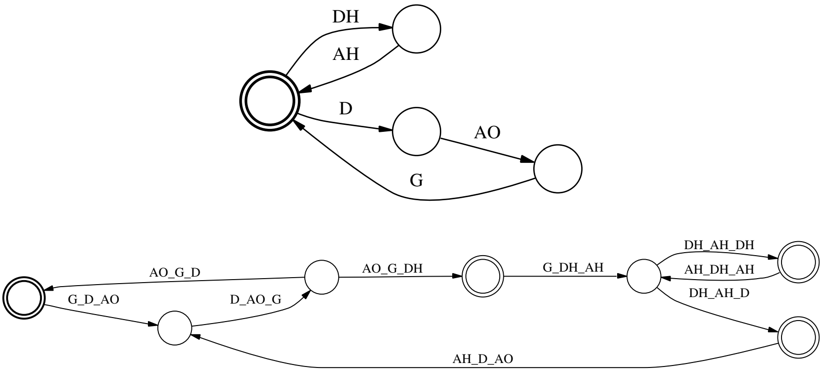


Context-Dependent Graph Expansion



- How can we do context-dependent expansion?
  - Handling branch points is tricky.
- Other tricky cases.
  - Words consisting of a single phone.
  - Quinphone models.

Triphone Graph Expansion Example

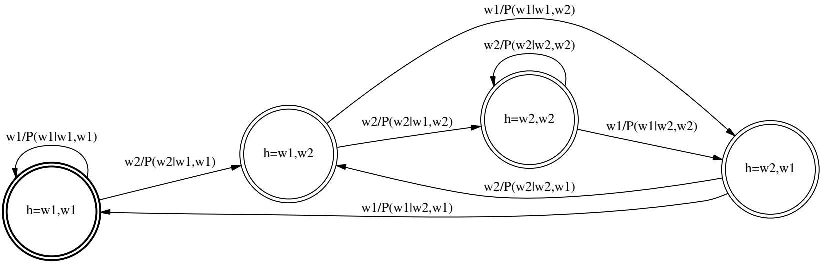


Aside: Word-Internal Acoustic Models

- Simplify acoustic model to simplify graph expansion.
- *Word-internal* models.
  - Don't let decision trees ask questions across word boundaries.
  - Pad contexts with the *unknown phone*.
  - Hurts performance (*e.g.*, coarticulation across words).
- As with word models, just replace each word with its HMM.

Issue: How Big The Graph?

- Trigram model (*e.g.*, vocabulary size $|V| = 2$)



- $|V|^3$ word arcs in FSA representation.
- Say words are $\sim$4 phones = 12 states on average.
- If $|V| = 50000$, $50000^3 \times 12 \approx 10^{15}$ states in graph.
- PC's have $\sim 10^9$ bytes of memory.

Issue: How Slow Decoding?

- In each frame, loop through every state in graph.
- If 100 frames/sec, 10^{15} states ...
  - How many cells to compute per second?
- PC's can do $\sim 10^{10}$ floating-point ops per second.

Recap: Small *vs.* Large Vocabulary Decoding

- In theory, can use the same exact techniques.
- In practice, three big problems:
  - (Context-dependent) graph expansion is complicated.
  - The decoding graph would be way too big.
  - Decoding would be way too slow.

Part II

Finite-State Transducers

A View of Graph Expansion

- Step 1: Take word graph as input.
  - Convert into phone graph.
- Step 2: Take phone graph as input.
  - Convert into context-dependent phone graph.
- Step 3: Take context-dependent phone graph.
  - Convert into HMM.

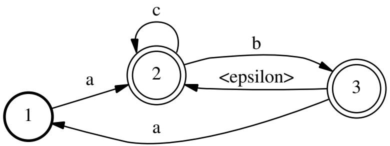
A Framework for Rewriting Graphs

- A general way of representing graph transformations?
  - Finite-state transducers (FST's).
- A general operation for applying transformations to graphs?
  - Composition.

Where Are We?

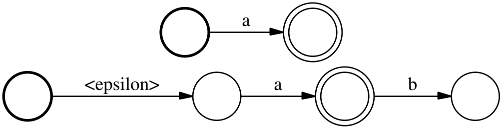
Review: What is a Finite-State Acceptor?

- It has states.
  - Exactly one initial state; one or more final states.
- It has arcs.
  - Each arc has a label, which may be empty (ϵ).
- Ignore probabilities for now.



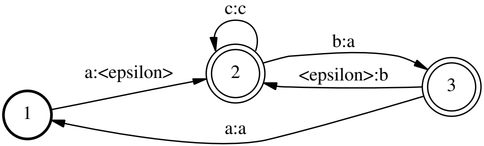
What Does an FSA *Mean*?

- The (possibly infinite) list of strings it accepts.
  - We need this in order to define composition.
- Things that *don't* affect meaning.
  - How labels are distributed along a path.
  - Invalid paths.
- Are these equivalent?



What is a Finite-State Transducer?

- It's like a finite-state acceptor, except ...
- Each arc has two labels instead of one.
  - An *input* label (possibly empty).
  - An *output* label (possibly empty).



What Does an FST *Mean*?

- A (possibly infinite) list of pairs of strings ...
  - An input string and an output string.
- The gist of *composition*.
  - If string $i_1 \cdots i_N$ occurs in input graph ...
  - And $(i_1 \cdots i_N, o_1 \cdots o_M)$ occurs in transducer, ...
  - Then string $o_1 \cdots o_M$ occurs in output graph.

Terminology

- *Finite-state acceptor* (FSA): one label on each arc.
- *Finite-state transducer* (FST): input and output label on each arc.
- *Finite-state machine* (FSM): FSA or FST.
  - Also, *finite-state automaton*.

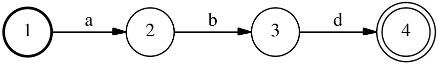
Where Are We?

The Composition Operation

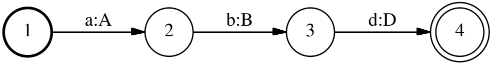
- A simple and efficient algorithm for computing ...
  - Result of applying a transducer to an acceptor.
- Composing FSA A with FST T to get FSA $A \circ T$.
  - If string $i_1 \cdots i_N \in A$ and ...
  - Input/output string pair $(i_1 \cdots i_N, o_1 \cdots o_M) \in T$, ...
  - Then string $o_1 \cdots o_M \in A \circ T$.

Rewriting a Single String A Single Way

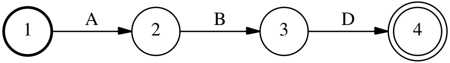
A



T

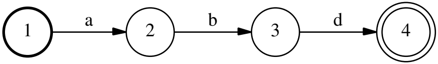


$A \circ T$

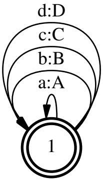


Rewriting a Single String A Single Way

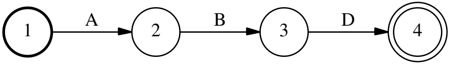
A



T



$A \circ T$



Transforming a Single String

- Let's say you have a string, *e.g.*,

  THE DOG

- Let's say we want to apply a one-to-one transformation.
  - *e.g.*, map words to their (single) baseforms.

  DH AH D AO G

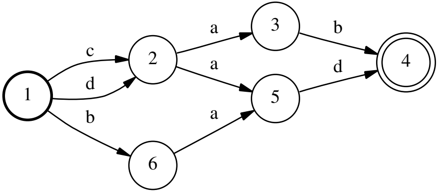
- This is easy, *e.g.*, use `sed` or `perl` or ...

The Magic of FST's and Composition

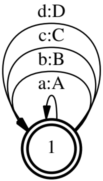
- Let's say you have a (possibly infinite) list of strings ...
  - Expressed as an FSA, as this is compact.
- How to transform all strings in FSA in one go?
- How to do one-to-many or one-to-zero transformations?
- Can we have the (possibly infinite) list of output strings ...
  - Expressed as an FSA, as this is compact?
- Fast?

Rewriting Many Strings At Once

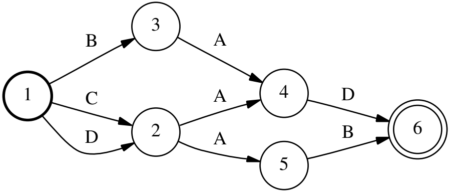
A



T

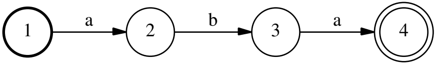


$A \circ T$

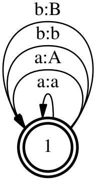


Rewriting A Single String Many Ways

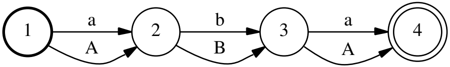
A



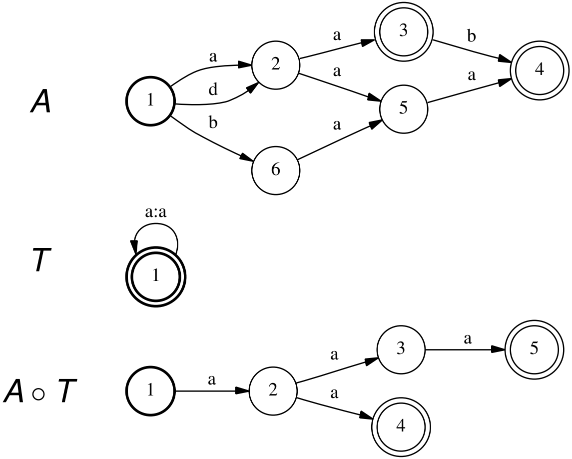
T



$A \circ T$



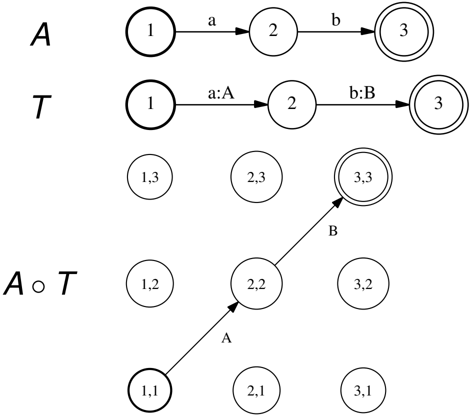
Rewriting Some Strings Zero Ways



Computing Composition: The Basic Idea

- For every state $s \in A$, $t \in T$, create state $(s, t) \in A \circ T$ ...
  - Corresponding to simultaneously being in states s and t.
- Make arcs in the intuitive way.

Example



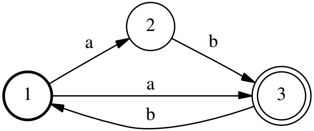
- Optimization: start from initial state, build outward.

Computing Composition: More Formally

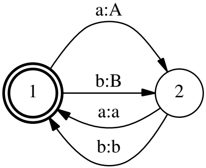
- For now, pretend no ϵ-labels.
- For every state $s \in A$, $t \in T$, create state $(s, t) \in A \circ T$.
- Create arc from (s_1, t_1) to (s_2, t_2) with label o iff ...
  - There is an arc from s_1 to s_2 in A with label i and ...
  - There is an arc from t_1 to t_2 in T with input label i and output label o.
- (s, t) is initial iff s and t are initial; similarly for final states.
- (Remove arcs and states that cannot reach both an initial and final state.)
- What is time complexity?

Another Example

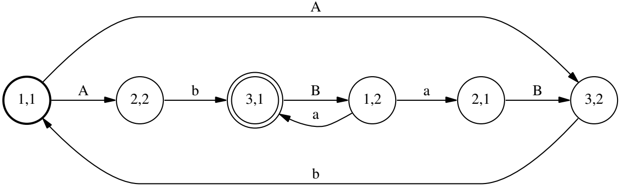
A



T



$A \circ T$

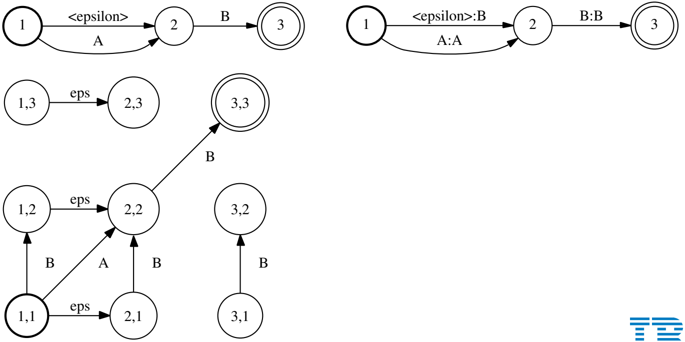


Composition and ϵ-Transitions

- Basic idea: can take ϵ-transition in one FSM without moving in other FSM.
  - A little tricky to do exactly right.
  - Do the readings if you care: (Pereira, Riley, 1997)

A, T

$A \circ T$



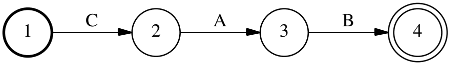
Recap: FST's and Composition

- Just as FSA's are a simple formalism that ...
  - Lets us express a large and interesting set of languages ...
- FST's are a simple formalism that ...
  - Lets us express a large and interesting set of one-to-many string transformations ...
- And the operation of composition lets us efficiently ...
  - Apply an FST to all strings in an FSA in one go!

FSM Toolkits

- AT&T FSM toolkit ⇒ OpenFST; lots of others.
  - Packages up composition, lots of other finite-state operations.
- A syntax for specifying FSA's and FST's, *e.g.*,

```
1   2   C
2   3   A
3   4   B
4
```



Where Are We?

1. What Is an FST?
2. Composition
3. FST's, Composition, and ASR
4. Weights

Graph Expansion: Original View

- Step 1: Take word graph as input.
  - Convert into phone graph.
- Step 2: Take phone graph as input.
  - Convert into context-dependent phone graph.
- Step 3: Take context-dependent phone graph.
  - Convert into HMM.

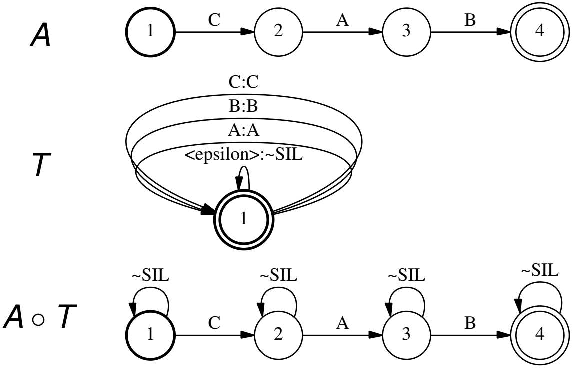
Graph Expansion: New View

- Final decoding graph: $L \circ T_1 \circ T_2 \circ T_3$.
  - L = language model FSA.
  - T_1 = FST mapping from words to CI phone sequences.
  - T_2 = FST mapping from CI phone sequences to CD phone sequences.
  - T_3 = FST mapping from CD phone sequences to GMM sequences.
- How to design T_1, T_2, T_3?

How To Design an FST?

- Design FSA accepting correct set of strings . . .
  - Keeping track of necessary “state”, *e.g.*, for CD expansion.
- Add in output tokens.
  - Creating additional states/arcs as necessary.

Example: Inserting Optional Silences



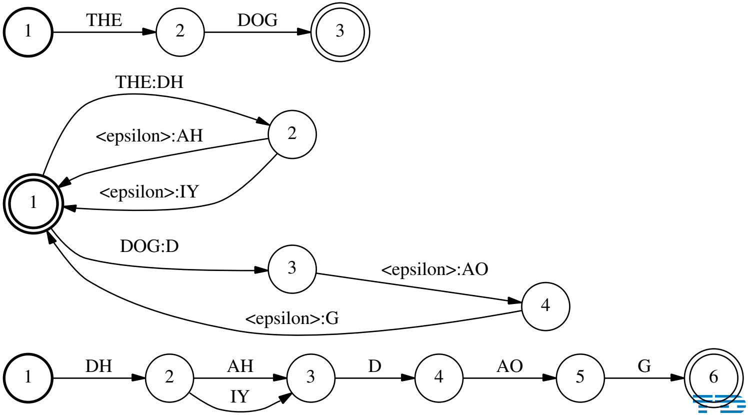
Example: Mapping Words To Phones

THE(01)	DH	AH
THE(02)	DH	IY

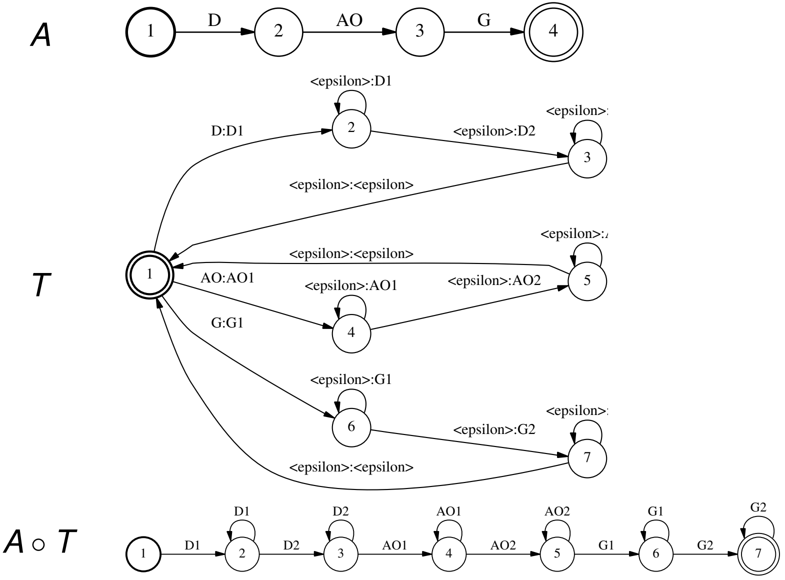
A

T

$A \circ T$



Example: Rewriting CI Phones as HMM's

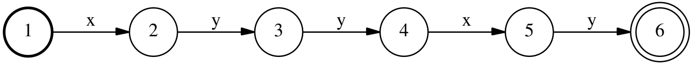


How to Express CD Expansion via FST's?

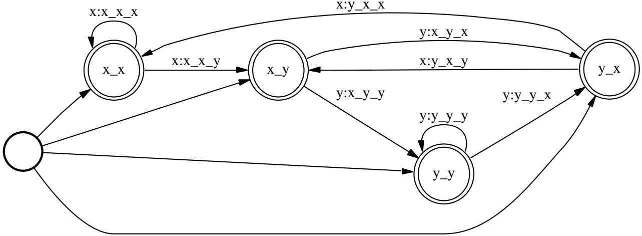
- Step 1: Rewrite each phone as a triphone.
  - Rewrite `AX` as `DH_AX_R` if `DH` to left, `R` to right.
  - One strategy: delay output of each phone by one arc.
  - What information to store in each state? (Think *n*-gram models.)
- Step 2: Rewrite each triphone with correct context-dependent HMM.
  - Just like rewriting a CI phone as its HMM.
  - Need to precompute HMM for each possible triphone.
  - See previous slide.

How to Express CD Expansion via FST's?

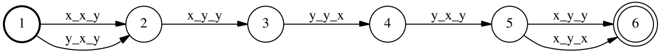
A



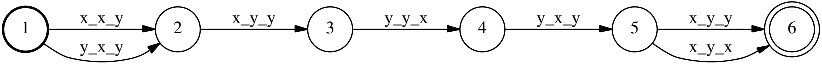
T



$A \circ T$



How to Express CD Expansion via FST's?



- Point: composition automatically expands FSA to correctly handle context!
  - Makes multiple copies of states in original FSA . . .
  - That can exist in different triphone contexts.
  - (And makes multiple copies of *only* these states.)

Quinphones and Beyond?

- Step 1: Rewrite each phone as a quinphone?
  - $50^5 \approx 300M$ arcs.
- Observation: given a word vocabulary ...
  - Not all quinphones can occur (usually).
- Build FST's to only handle quinphones that can occur.

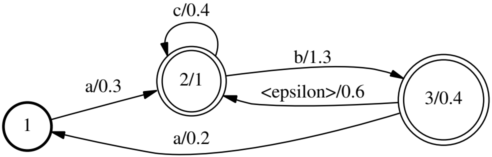
Recap: FST's and ASR

- Graph expansion can be framed as series of composition operations.
- Building the FST's for each step is pretty straightforward ...
  - Except for context-dependent phone expansion.
- Once you have the FST's, easy peasy.
  - Composition handles context-dependent expansion correctly.
- Handles graph expansion for training, too.

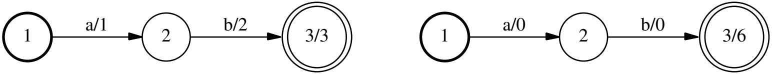
Where Are We?

What About Those Probability Thingies?

- *e.g.*, to hold language model probs, transition probs, etc.
- FSM's ⇒ *weighted* FSM's.
  - WFSA's, WFST's.
- Each arc has a score or *cost*.
  - So do final states.

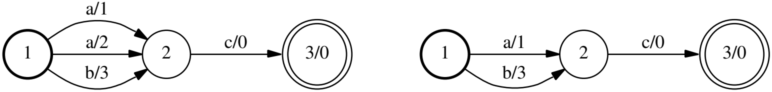


- The (possibly infinite) list of strings it accepts . . .
  - And for each string, a cost.
- Typically, we take costs to be negative log probabilities.
  - Cost of a path is sum of arc costs plus final cost.
  - (Total path log prob is sum of arc log probs.)
- Things that *don't* affect meaning.
  - How costs or labels are distributed along a path.
  - Invalid paths.
- Are these equivalent?



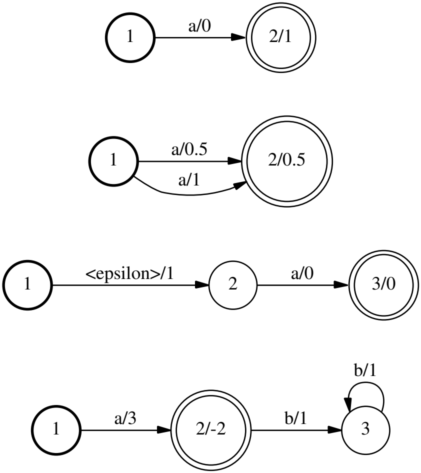
What If Two Paths With Same String?

- How to compute cost for this string?
- Use min operator to compute combined cost (Viterbi)?
  - Can combine paths with same labels without changing meaning.



- Operations $(+, \min)$ form a *semiring* (the *tropical* semiring).
  - Other semirings are possible.

Which Is Different From the Others?

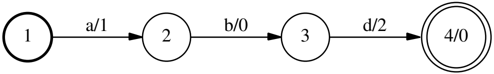


Weighted Composition

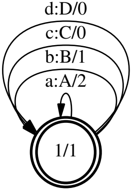
- If $(i_1 \cdots i_N, c)$ in input graph ...
- And $(i_1 \cdots i_N, o_1 \cdots o_M, c')$ in transducer, ...
- Then $(o_1 \cdots o_M, c + c')$ in output graph.
- Combine costs for all different ways to produce same $o_1 \cdots o_M$.

Weighted Composition

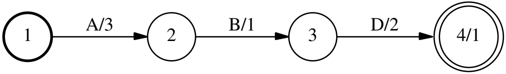
A



T



$A \circ T$



Weighted Composition and ASR

$$\begin{aligned}
\text{class}(\mathbf{x}) &= \arg\max_{\omega} \; P(\omega)P(\mathbf{x}|\omega) \\
P(\mathbf{x}|\omega) &\approx \max_{A} \prod_{t=1}^{T} P(a_t) \prod_{t=1}^{T} P(\vec{x}_t|a_t) \\
P(\omega = w_1 \cdots w_l) &= \prod_{i=1}^{l+1} P(w_i|w_{i-2}w_{i-1})
\end{aligned}$$

- Total log prob of path is sum over component log probs.
- In Viterbi, if multiple paths labeled with same string ...
- Only pay attention to path with highest log prob.

Weighted Composition and ASR

- ASR decoding.
  - Total log prob of path is sum over component log probs.
  - In Viterbi, if multiple paths labeled with same string . . .
  - Only pay attention to path with highest log prob.
- Weighted FSM's; cost = negative log prob.
  - Total cost of path is sum of costs on arcs.
  - If multiple paths labeled with same string . . .
  - Only pay attention to path with lowest cost.
  - Weighted composition sums costs from input machines.

The Bottom Line

- Final decoding graph: $L \circ T_1 \circ T_2 \circ T_3$.
  - L = language model FSA.
  - T_1 = FST mapping from words to CI phone sequences.
  - T_2 = FST mapping from CI phone sequences to CD phone sequences.
  - T_3 = FST mapping from CD phone sequences to GMM sequences.
- If put component LM, AM log probs in L, T_1, T_2, T_3, ...
  - Then doing Viterbi decoding on $L \circ T_1 \circ T_2 \circ T_3 \ldots$
  - Will correctly compute:

$$\text{class}(\mathbf{x}) = \arg\max_{\omega} P(\omega) P(\mathbf{x}|\omega)$$

Weighted Graph Expansion

- Final decoding graph: $L \circ T_1 \circ T_2 \circ T_3$.
  - L = language model FSA (w/ LM costs).
  - T_1 = FST mapping from words to CI phone sequences (w/ pronunciation costs).
  - T_2 = FST mapping from CI phone sequences to CD phone sequences.
  - T_3 = FST mapping from CD phone sequences to GMM sequences (w/ HMM transition costs).
- In final graph, each path has correct "total" cost.

Recap: Weighted FSM's and ASR

- Graph expansion can be framed as series of composition operations . . .
  - Even when you need to worry about probabilities.
- Weighted composition correctly combines scores from multiple WFSM's.
- Varying the semiring used can give you other behaviors.
  - *e.g.*, can we sum probs across paths rather than max?

Recap: FST's and Composition

- Like `sed`, but can operate on all paths in a lattice simultaneously.
- Rewrite symbols as other symbols.
  - *e.g.*, rewrite words as phone sequences (or vice versa).
- Context-dependent rewriting of symbols.
  - *e.g.*, rewrite CI phones as their CD variants.
- Add in new scores.
  - *e.g.*, language model lattice rescoring.
- Restrict the set of allowed paths/intersection.
  - *e.g.*, find all paths in lattice containing word NOODGE.
- Or all of the above at once.

Part III

Making Decoding Efficient

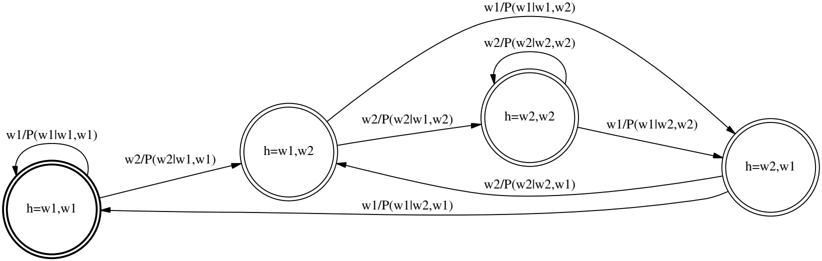
The Problem

- Naive graph expansion, trigram LM.
  - If $|V| = 50000$, $50000^3 \times 12 \approx 10^{15}$ states in graph.
- Naive Viterbi on this graph.
  - 10^{15} states $\times$ 100 frames/sec = 10^{17} cells/sec.
- Two main approaches.
  - Reduce states in graph: saves memory and time.
  - Don't process all cells in chart.

Where Are We?

Compactly Representing *N*-Gram Models

- For trigram model, $|V|^2$ states, $|V|^3$ arcs in naive representation.



- Only a small fraction of the possible $|V|^3$ trigrams will occur in the training data.
  - Is it possible to keep arcs only for occurring trigrams?

Compactly Representing N-Gram Models

- Can express smoothed n-gram models via backoff distributions

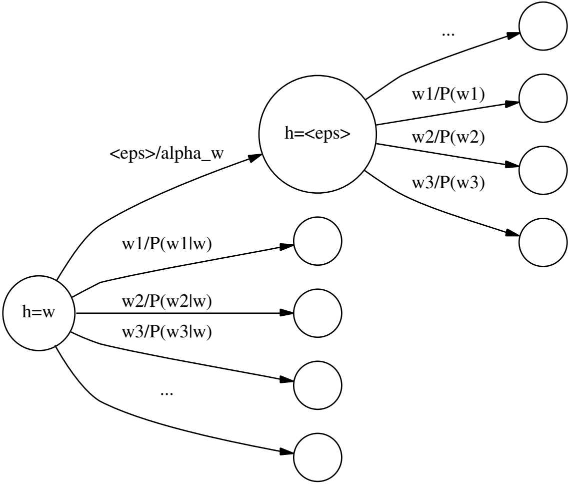
$$P_{\text{smooth}}(w_i | w_{i-1}) = \begin{cases} P_{\text{primary}}(w_i | w_{i-1}) & \text{if count}(w_{i-1} w_i) > 0 \\ \alpha_{w_{i-1}} P_{\text{smooth}}(w_i) & \text{otherwise} \end{cases}$$

- *e.g.*, Witten-Bell smoothing

$$\begin{aligned} P_{\text{WB}}(w_i | w_{i-1}) \quad = \quad & \frac{c_h(w_{i-1})}{c_h(w_{i-1}) + N_{1+}(w_{i-1})} P_{\text{MLE}}(w_i | w_{i-1}) + \\ & \frac{N_{1+}(w_{i-1})}{c_h(w_{i-1}) + N_{1+}(w_{i-1})} P_{\text{WB}}(w_i) \end{aligned}$$

Compactly Representing *N*-Gram Models

$$P_{\text{smooth}}(w_i|w_{i-1}) = \begin{cases} P_{\text{primary}}(w_i|w_{i-1}) & \text{if } \text{count}(w_{i-1}w_i) > 0 \\ \alpha_{w_{i-1}} P_{\text{smooth}}(w_i) & \text{otherwise} \end{cases}$$



Compactly Representing *N*-Gram Models

- By introducing backoff states ...
  - Only need arcs for *n*-grams with nonzero count.
  - Compute probabilities for *n*-grams with zero count ...
  - By traversing backoff arcs.
- Does this representation introduce any error?
  - Hint: are there multiple paths with same label sequence?

Can We Make the LM Even Smaller?

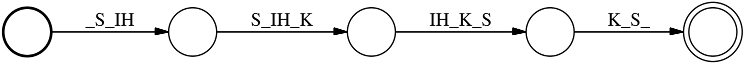
- Sure, just remove some more arcs. Which?
- Count cutoffs.
  - *e.g.*, remove all arcs corresponding to bigrams . . .
  - Occurring fewer than k times in the training data.
- Likelihood/entropy-based pruning.
  - Choose those arcs which when removed, . . .
  - Change the likelihood of the training data the least.
  - (Seymore and Rosenfeld, 1996), (Stolcke, 1998)

LM Pruning and Graph Sizes

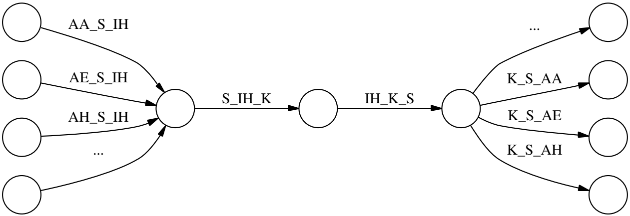
- Original: trigram model, $|V|^3 = 50000^3 \approx 10^{14}$ word arcs.
- Backoff: $>$100M unique trigrams $\Rightarrow$ $\sim$100M word arcs.
- Pruning: keep $<$5M *n*-grams $\Rightarrow$ $\sim$5M word arcs.
  - 4 phones/word $\Rightarrow$ 12 states/word $\Rightarrow$ $\sim$60M states?
- We're done?

What About Context-Dependent Expansion?

- With word-internal models, each word really is only ∼12 states



- With cross-word models, each word is hundreds of states?
  - 50 CD variants of first three states, last three states.



Where Are We?

5 Shrinking N-Gram Models

6 Graph Optimization

7 Pruning Search

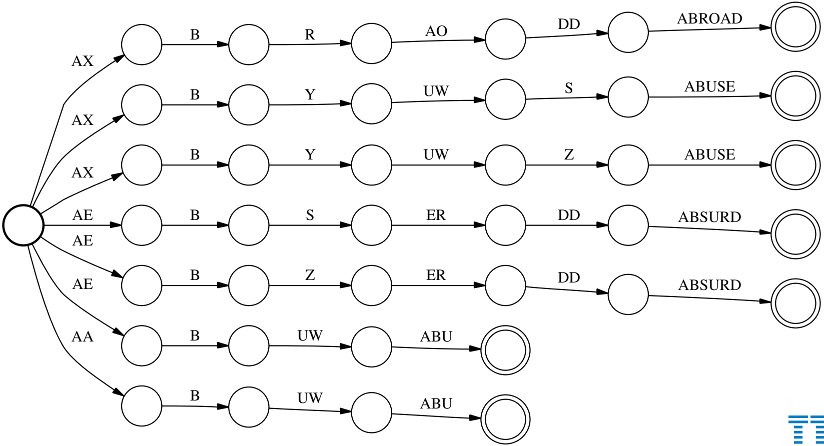
8 Saving Memory

Graph Optimization

- Can we modify the topology of a graph ...
  - Such that it's smaller (fewer arcs or states) ...
  - Yet retains the same *meaning*.
- The meaning of an WFSA:
  - The set of strings it accepts, and the cost of each string.
  - Don't care how costs or labels are distributed along a path.

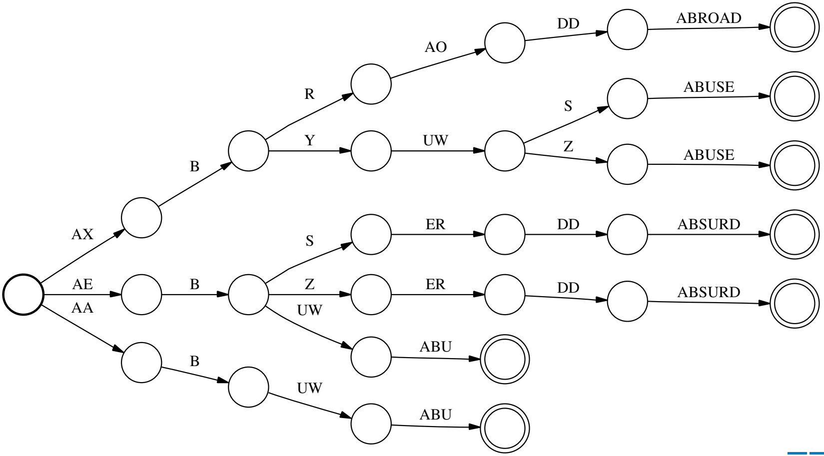
Graph Compaction

- Consider word graph for isolated word recognition.
  - Expanded to phone level: 39 states, 38 arcs.



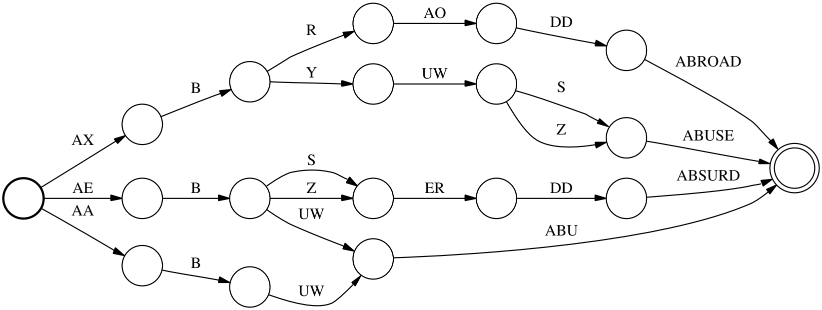
Determinization

- Share common prefixes: 29 states, 28 arcs.



Minimization

- Share common suffixes: 18 states, 23 arcs.

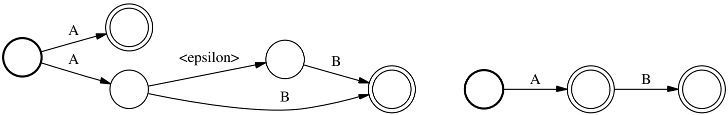


Determinization and Minimization

- By sharing arcs between paths . . .
  - We reduced size of graph by half . . .
  - Without changing its meaning.
- *determinization* — prefix sharing.
  - Produce *deterministic* version of an FSM.
- *minimization* — suffix sharing.
  - Given a deterministic FSM, find equivalent FSM with minimal number of states.

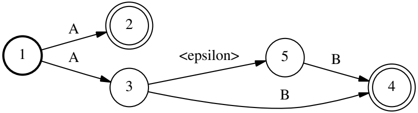
What Is A Deterministic FSM?

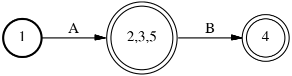
- No two arcs exiting the same state have the same input label.
- No ϵ arcs.
- *i.e.*, for any input label sequence ...
  - At most one path from start state labeled with that sequence.



Determinization: The Basic Idea

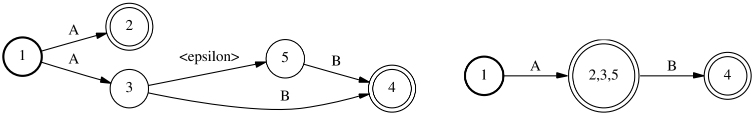
- For an input label sequence ...
  - There is set of states you can reach from start state ...
  - Accepting exactly that input sequence.
- Collect all such state sets (over all input sequences).
  - Each such state set maps to a state in the output FSM.
- Make arcs in the logical way.



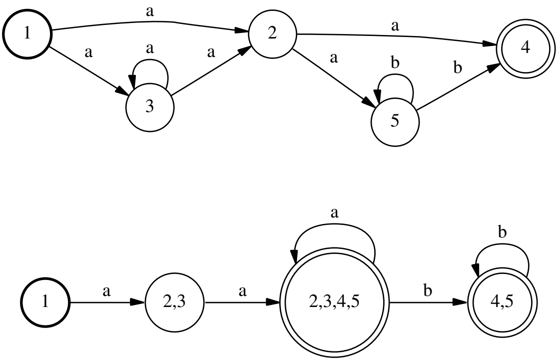


Determinization

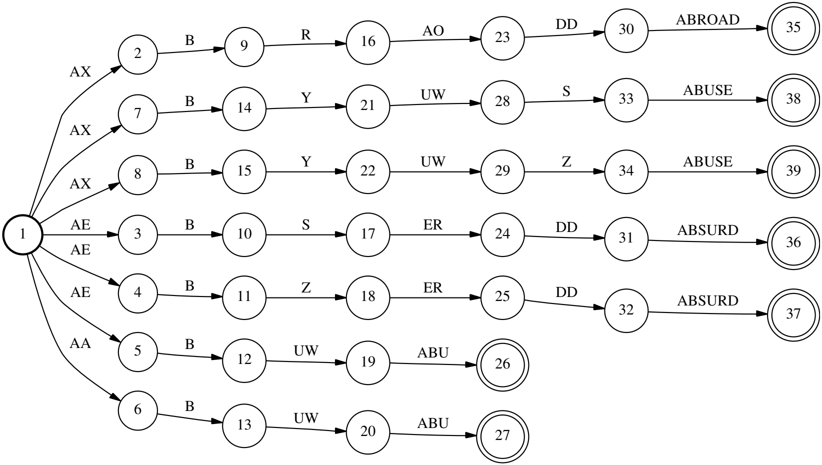
- Start from start state.
- Keep list of state sets not yet expanded.
    - For each, find outgoing arcs, creating new state sets as needed.
- Must follow ϵ arcs when computing state sets.



Example 2

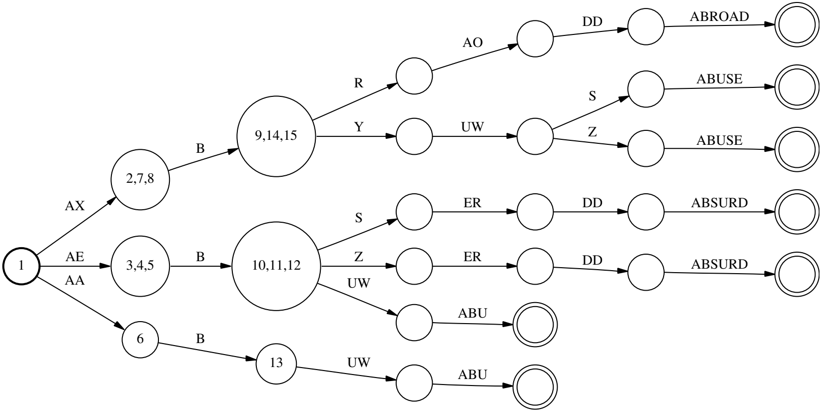


Example 3



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
AX
AX
AX
AE
AE
AE
AA
B
R
AO
DD
ABROAD
Y
UW
S
ABUSE
Z
ER
ABSURD
ABU

Example 3, Continued



Pop Quiz: Determinization

- Are all unweighted FSA's determinizable?
  - *i.e.*, will the determinization algorithm always terminate?
- For an FSA with s states, ...
  - What is the maximum number of states in its determinization?

Recap: Determinization

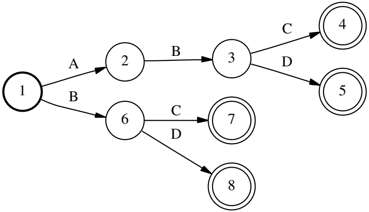
- Improves behavior of composition and search!
  - In composition, output states (s, t) created when?
- Whether reduces or increases number of states ...
  - Depends on nature of input FSM.
- Required for minimization algorithm.
- Can apply to weighted FSM's and transducers as well.

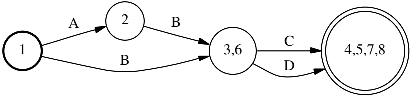
Minimization

- Given a deterministic FSM ...
  - Find equivalent deterministic FSM with minimal number of states.
- Number of arcs may be nowhere near minimal.
  - Minimizing number of arcs is NP-complete

Minimization: Acyclic Graphs

- Merge states with same following strings (*follow sets*).





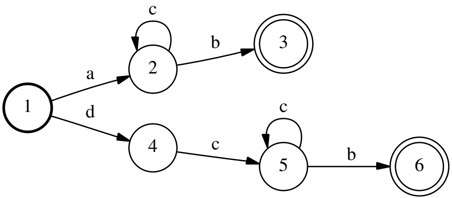
states	following strings
1	ABC, ABD, BC, BD
2	BC, BD
3, 6	C, D
4,5,7,8	ϵ

General Minimization: The Basic Idea

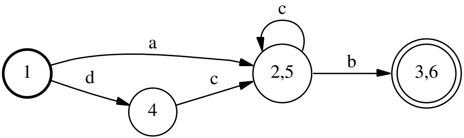
- Start with all states in single partition.
- Whenever find evidence that two states within partition . . .
  - Have different follow sets . . .
  - Split the partition.
- At end, collapse all states in same partition into single state.

Minimization

- Invariant: if two states are in different partitions . . .
  - They have different follow sets.
  - Converse does not hold.
- First split: final and non-final states.
  - Final states have ϵ in their follow sets; non-final states do not.
- If two states in same partition have . . .
  - Different number of outgoing arcs, or different arc labels . . .
  - Or arcs go to different partitions . . .
  - The two states have different follow sets.



action	evidence	partitioning
		{1,2,3,4,5,6}
split 3,6	final	{1,2,4,5}, {3,6}
split 1	has *a* arc	{1}, {2,4,5}, {3,6}
split 4	no *b* arc	{1}, {4}, {2,5}, {3,6}



Recap: Minimization

- Minimizes states, not arcs, for deterministic FSM's.
- Does minimization always terminate?
- Not that expensive, can sometimes get something.
- Can apply to weighted FSM's and transducers as well.
  - Need to first apply *push* operation.
  - Normalizes locations of costs/labels along paths . . .
  - So arcs that can be merged will have same cost/label.
- Determinization and minimization available in FSM toolkits.

Weighted Graph Expansion, Optimized

- Final decoding graph: min(det($L \circ T_1 \circ T_2 \circ T_3$)).
  - L = pruned, backoff language model FSA.
  - T_1 = FST mapping from words to CI phone sequences.
  - T_2 = FST mapping from CI phone sequences to CD phone sequences.
  - T_3 = FST mapping from CD phone sequences to GMM sequences.
- 10^{15} states $\Rightarrow$ 10–20M states/arcs.
  - 2–4M *n*-grams kept in LM.

Practical Considerations

- Final decoding graph: $\text{min}(\text{det}(L \circ T_1 \circ T_2 \circ T_3))$.
- Strategy: build big graph, then minimize at the end?
  - Problem: can't hold big graph in memory.
- Another strategy: minimize graph after each expansion step.
- A little bit of art involved.
  - Composition is associative.
  - Many existing recipes for graph expansion.

Historical Note

- In the old days (pre-AT&T):
  - People determinized their decoding graphs . . .
  - And did the push operation for LM lookahead . . .
  - Without calling it determinization or pushing.
  - ASR-specific implementations.
- Nowadays (late 1990's–)
  - FSM toolkits implementing general finite-state operations.
  - Can apply finite-state operations in many contexts in ASR.

Where Are We?

Real-Time Decoding

- Why is this desirable?
- Decoding time for Viterbi algorithm; 10M states in graph.
  - In each frame, loop through every state in graph.
  - 100 frames/sec $\times$ 10M states $\times$ $\sim$100 cycles/state $\Rightarrow$ 10^{11} cycles/sec.
  - PC's do $\sim$ 10^9 cycles/second (*e.g.*, 3GHz P4).
- We cannot afford to evaluate each state at each frame.
  - $\Rightarrow$ Pruning!

Pruning

- At each frame, only evaluate states/cells with best Viterbi scores.
- Given *active* states/cells from last frame . . .
  - Only examine states/cells in current frame . . .
  - Reachable from active states in last frame.
  - Keep best to get active states in current frame.

Pruning

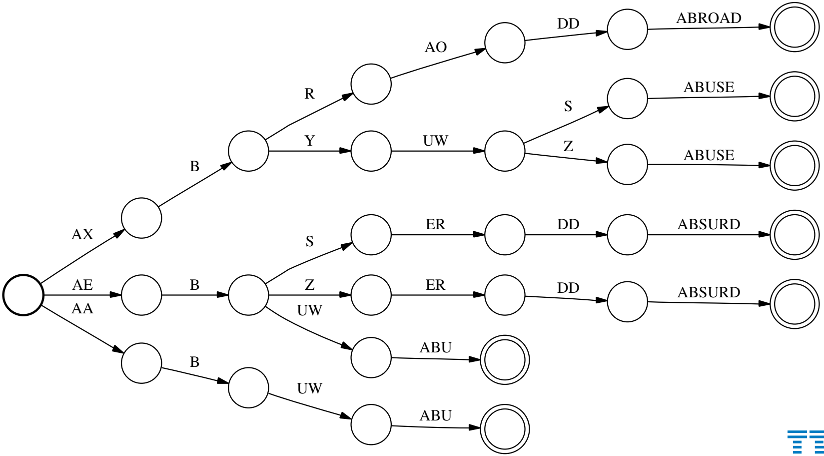
- When not considering every state at each frame . . .
  - We may make *search errors*.
- The field of *search* in ASR.
  - Trying to minimize computation *and* search errors.

How Many Active States To Keep?

- Goal: Try to prune paths . . .
  - With no chance of ever becoming the *best* path.
- *Beam* pruning.
  - Keep only states with log probs within fixed distance . . .
  - Of best log prob at that frame.
  - Why does this make sense? When could this be bad?
- *Rank* or *histogram* pruning.
  - Keep only k highest scoring states.
  - Why does this make sense? When could this be bad?
- Can we get the best of both worlds?

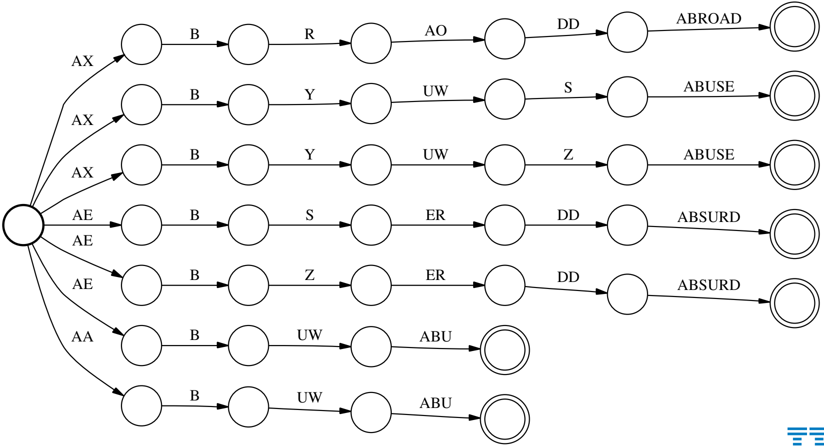
Pruning Visualized

- Active states are small fraction of total states (<1%)
- Tend to be localized in small regions in graph.



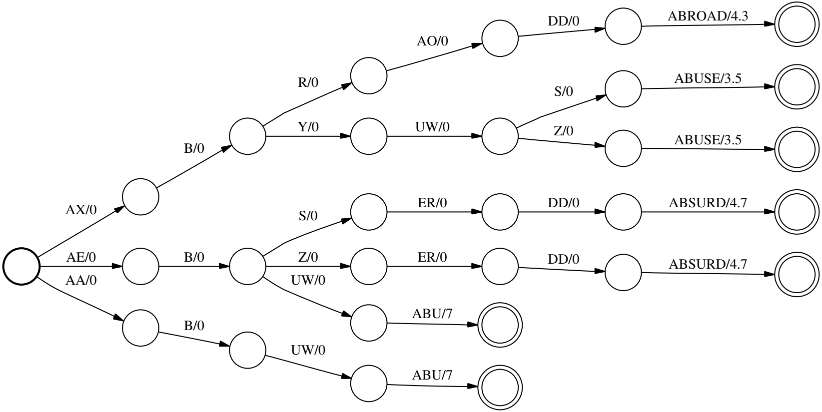
Pruning and Determinization

- Most uncertainty occurs at word starts.
- Determinization drastically reduces branching here.



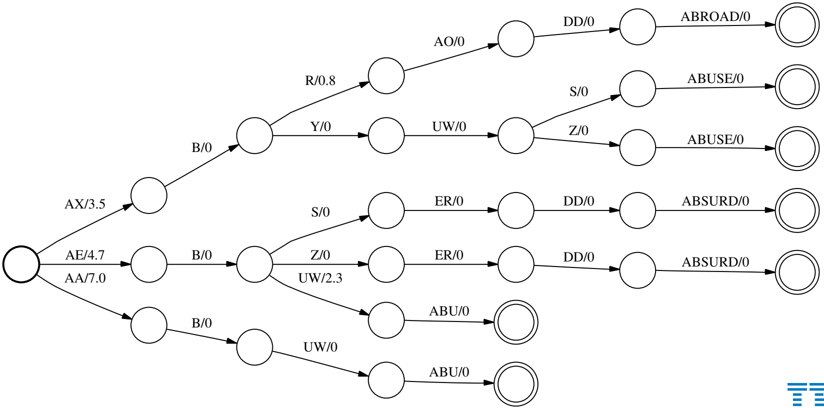
Language Model Lookahead

- In practice, put word labels at word ends. (Why?)
- What's wrong with this picture? (Hint: think beam pruning.)



Language Model Lookahead

- Move LM scores as far ahead as possible.
- At each point, total cost ⇔ min LM cost of following words.
- *push* operation does this.



Recap: Efficient Viterbi Decoding

- Pruning is key.
- Pruning behavior improves immensely with . . .
  - Determinization.
  - LM lookahead.
- Can process $\sim$10000 states/frame in $<$ 1x RT on a PC.
  - Can process $\sim$1% of cells for 10M-state graph . . .
  - And make very few search errors.
- Can go even faster with smaller LM's (or more search errors).

Where Are We?

5 Shrinking N-Gram Models

6 Graph Optimization

7 Pruning Search

8 Saving Memory

What's the Problemo?

- Naive implementation: store whole DP chart.
- If 10M-state decoding graph:
  - 10 second utterance $\Rightarrow$ 1000 frames.
  - 1000 frames $\times$ 10M states = 10 billion cells in DP chart.
- Each cell holds:
  - Viterbi log prob.
  - Backtrace pointer.

Optimization 1: Sparse Chart

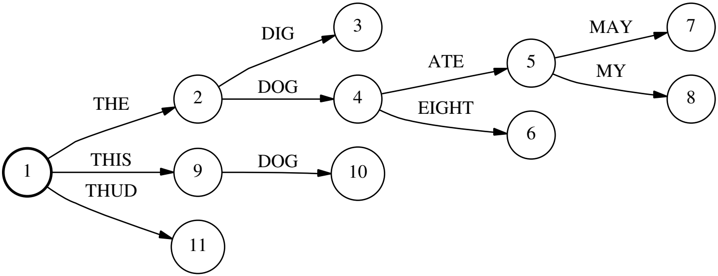
- Use sparse representation of DP chart.
  - Only store cells for *active* states.
- 10M cells/frame $\Rightarrow$ 10k cells/frame.

Optimization 2: Forgetting the Past

- Insight: the only reason we need to keep around cells from past frames . . .
  - Is so we can do backtracing to recover the final word sequence.
- Can we store backtracing information in some other way?

Token Passing

- Maintain "word tree":
  - Compact encoding of a list of similar word sequences.
- Backtrace pointer points to node in tree ...
  - Holding word sequence labeling best path to cell.
- Set backtrace to same node as at best last state ...
  - Unless cross word boundary.



Recap: Saving Memory in Viterbi Decoding

- Before:
  - Static decoding graph.
  - (# states) $\times$ (# frames) cells.
- After:
  - Static decoding graph (shared memory) $\Leftarrow$ the biggie.
  - (# active states) $\times$ (2 frames) cells.
  - Backtrace word tree.

Part IV

Other Decoding Paradigms

Where Are We?

My Graph Is Too Big

- One approach: *static graph expansion*.
  - Shrink the graph by . . .
  - Using a simpler language model and . . .
  - Statically optimizing the graph.
- Another approach: *dynamic graph expansion*.
  - Don't store the whole graph in memory.
  - Build the parts of the graph with active states on the fly.

A Tale of Two Decoding Styles

- Approach 1: Dynamic graph expansion.
  - Since late 1980's.
  - Can handle more complex language models.
  - Decoders are incredibly complex beasts.
  - *e.g.*, cross-word CD expansion without FST's.
- Approach 2: Static graph expansion.
  - Pioneered by AT&T in late 1990's.
  - Enabled by optimization algorithms for WFSM's.
  - Static graph expansion is complex.
  - Decoding is relatively simple.

Dynamic Graph Expansion

- How can we store a really big graph such that ...
  - It doesn't take that much memory, but ...
  - Easy to expand any part of it that we need.
- Observation: composition is associative:

$$(A \circ T_1) \circ T_2 = A \circ (T_1 \circ T_2)$$

- Observation: decoding graph is composition of LM with a bunch of FST's:

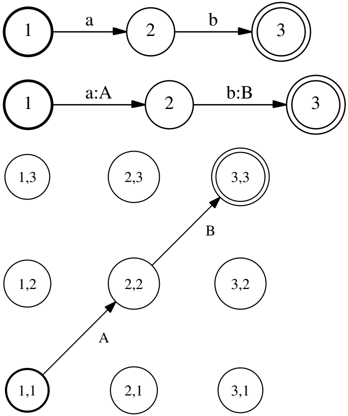
$$\begin{aligned} G_{\text{decode}} &= A_{\text{LM}} \circ T_{\text{wd}\to\text{pn}} \circ T_{\text{CI}\to\text{CD}} \circ T_{\text{CD}\to\text{HMM}} \\ &= A_{\text{LM}} \circ (T_{\text{wd}\to\text{pn}} \circ T_{\text{CI}\to\text{CD}} \circ T_{\text{CD}\to\text{HMM}}) \end{aligned}$$

Review: Composition

A

T

$A \circ T$



On-the-Fly Composition

$$G_{\text{decode}} = A_{\text{LM}} \circ (T_{\text{wd}\to\text{pn}} \circ T_{\text{CI}\to\text{CD}} \circ T_{\text{CD}\to\text{HMM}})$$

- Instead of storing one big graph $G_{\text{decode}}, \ldots$
  - Store two smaller graphs: A_{LM} and $T = T_{\text{wd}\to\text{pn}} \circ T_{\text{CI}\to\text{CD}} \circ T_{\text{CD}\to\text{HMM}}$.
- Replace states with state *pairs* (s_A, s_T).
  - Straightforward to compute outgoing arcs of (s_A, s_T).

Notes: Dynamic Graph Expansion

- Really complicated to explain before FSM perspective.
- Other decompositions into component graphs are possible.
- Speed:
  - Statically optimize component graphs.
  - Try to approximate static optimization of composed graph . . .
  - Using on-the-fly techniques.

Where Are We?

Synchronicity

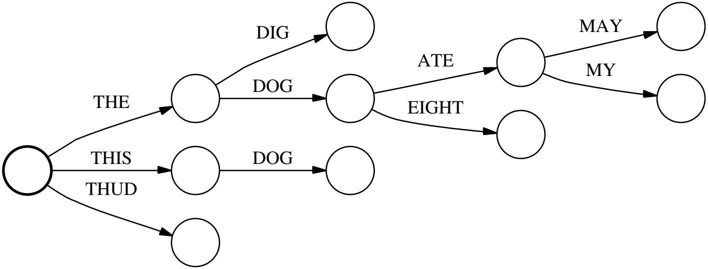
- Synchronous search — *e.g.*, Viterbi search.
  - Extend all paths and calculate all scores synchronously.
  - Expand states with mediocre scores in case improve later.
- Asynchronous search — *e.g.*, stack search.
  - Pursue best-looking path first, regardless of length!
  - If lucky, expand very few states at each frame.

Stack Search

- Pioneered at IBM in mid-1980's; first real-time dictation system.
- May be competitive at low-resource operating points; low noise.
    - Difficult to tune (nonmonotonic behavior w.r.t. parameters).
    - Going out of fashion?

Stack Search

- Extend hypotheses word-by-word
- Use *fast match* to decide which word to extend best path with.
  - Decode single word with simpler acoustic model.



Stack Search

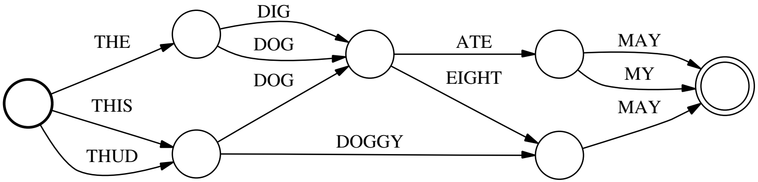
- Advantages.
  - If best path pans out, very little computation.
- Disadvantages.
  - Difficult to compare paths of different lengths.
  - May need to recompute the same values multiple times.

Where Are We?

Two-Pass Decoding

- What about my fuzzy logic 15-phone acoustic model and 7-gram neural net language model with SVM boosting?
- Some of the ASR models we develop in research are ...
  - Too expensive to implement in one-pass decoding.
- First-pass decoding: use simpler model ...
  - To find “likeliest” word *sequences* ...
  - As lattice (WFSA) or flat list of hypotheses (*N*-best list).
- *Rescoring*: use complex model ...
  - To find best word sequence from among first-pass hypotheses.

Lattice Generation and Rescoring



- In Viterbi, store *k*-best tracebacks at each word-end cell.
- To add in new LM scores to a lattice . . .
  - What operation can we use?
- Lattices have other uses.
  - *e.g.*, confidence estimation, consensus decoding, lattice MLLR, etc.

N-Best List Rescoring

- For exotic models, even lattice rescoring may be too slow.
  - For some models, computation linear in number of hypotheses.
- Easy to generate N-best lists from lattices.
  - A^* algorithm.
- N-best lists have other uses.
  - *e.g.*, confidence estimation, alternatives in interactive apps, etc.

Where Are We?

Synchronous or Asynchronous?

- Stack search: lots of search errors in noise.
- Only consider if very low memory footprint.

Static or Dynamic? Two-Pass?

- If speed is a premium?
- If flexibility is a premium?
  - *e.g.*, update LM vocabulary every night.
- If need a gigantic language model?
- If latency is a premium?
  - What can't we use?
- If accuracy is a premium (speed OK, no latency requirements)?
- If accuracy is a premium (all the time in the world)?
- If doing cutting-edge research?

The Road Ahead

- Weeks 1–4: Small vocabulary ASR.
- Weeks 5–8: Large vocabulary ASR.
- Weeks 9–12: Advanced topics.
  - Adaptation; robustness.
  - Advanced language modeling.
  - Discriminative training; ROVER; consensus.
  - Applications: ???.
- Week 13: Final presentations.

Course Feedback

1. Was this lecture mostly clear or unclear? What was the muddiest topic?
2. Other feedback (pace, content, atmosphere)?