INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Sampling from Stochastic Finite Automata with Applications to CTC Decoding

Martin Jansche, Alexander Gutkin

Google Research, London, United Kingdom

{mjansche,agutkin}@google.com

Abstract

Stochastic finite automata arise naturally in many language and speech processing tasks. They include stochastic acceptors, which represent certain probability distributions over random strings. We consider the problem of efficient sampling: drawing random string variates from the probability distribution represented by stochastic automata and transformations of those. We show that path-sampling is effective and can be efficient if the epsilon-graph of a finite automaton is acyclic. We provide an algorithm that ensures this by conflating epsilon-cycles within strongly connected components. Sampling is also effective in the presence of non-injective transformations of strings. We illustrate this in the context of decoding for Connectionist Temporal Classification (CTC), where the predictive probabilities yield auxiliary sequences which are transformed into shorter labeling strings. We can sample efficiently from the transformed labeling distribution and use this in two different strategies for finding the most probable CTC labeling.

Index Terms: finite-state techniques, stochastic automata, sampling, decoding

1. Foundations

Stochastic finite automata are used in many language and speech processing tasks, including speech recognition [1, 2] and synthesis [3, 4], language [5] and pronunciation [6] modeling, tagging [7] and parsing [8], input methods [9], among many others. They arise naturally in optimization and decoding tasks for probabilistic sequence models.

1.1. FST background

A weighted finite state transducer (WFST) is a tuple $(\Gamma, \Delta, S, Q, q_0, f, E, w)$ where Γ and Δ are finite sets (the input and output alphabet, respectively), $(S, \oplus, \otimes)$ is a semiring of weights, Q is a finite set of states, $q_0 \in Q$ is the initial state, $f : Q \to S$ is a function mapping a state to its final weight, $E \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times Q$ is a finite set of edges, and $w : E \to S$ is a function mapping an edge to its weight. Given an edge $e = (s, i, o, t)$ define canonical projections $\text{src}(e) = s$, $\text{istr}(e) = i$, $\text{ostr}(e) = o$, and $\text{tgt}(e) = t$.

A path $\pi = (e_1, \ldots, e_n)$ is a finite sequence of consecutive edges such that $\text{tgt}(e_i) = \text{src}(e_{i+1})$ for $i \in \{1, \ldots, n-1\}$. In this case we write $\text{src}(\pi) = \text{src}(e_1)$ and $\text{tgt}(\pi) = \text{tgt}(e_n)$, and we extend the weight function to paths:

$$w(\pi) = \left[\bigotimes_{i=1}^{n} w(e_i)\right] \otimes f(\text{tgt}(\pi)) .$$

Associated with a path $\pi = (e_1, \ldots, e_n)$ is an input string $\text{istr}(\pi) = \text{istr}(e_1) \cdots \text{istr}(e_n)$ as well as a similarly defined output string. The set of all successful paths labeled with given strings $u \in \Gamma^*$ and $v \in \Delta^*$ is $\text{Paths}(u, v) = \{\pi \mid \text{istr}(\pi) = u \wedge \text{ostr}(\pi) = v \wedge \text{src}(\pi) = q_0\}$. The behavior of the WFST $\mathscr{A}$ is the function

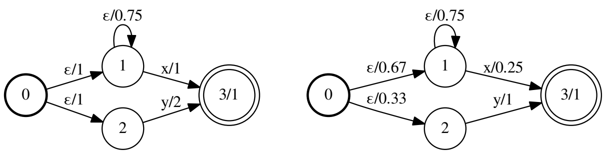

Figure 1: *Unnormalized and equivalent normalized SFST. On the left, the grand total weight is* 6*; on the right,* 1.

$[\![\mathscr{A}]\!] : \Gamma^* \times \Delta^* \to S$, and its grand total weight is $w(\mathscr{A})$:

$$[\![\mathscr{A}]\!](u,v) = \bigoplus_{\pi \in \text{Paths}(u,v)} w(\pi) \qquad w(\mathscr{A}) = \bigoplus_{(u,v) \in \Gamma^* \times \Delta^*} [\![\mathscr{A}]\!](u,v) . \tag{1}$$

A stochastic FST (SFST) is a WFST defined over the semiring of nonnegative real numbers $(\mathbb{R}^{\geq 0}, +, \cdot)$. An SFST $\mathscr{A}$ is globally normalized if $w(\mathscr{A}) = 1$; in that case $[\![\mathscr{A}]\!]$ is a probability measure for the countable sample space $\Gamma^* \times \Delta^*$. Normalization is impossible if the grand total diverges, due to cycles with path weights ≥ 1. An SFST is locally normalized if

$$\forall q \in Q \quad f(q) \oplus \bigoplus_{\substack{e \in E \\ \text{src}(e) = q}} w(e) = 1 .$$

An SFST can be brought into locally normalized form (implying global normalization) using the WFST weight pushing algorithm [10, pp. 241–43]. This is illustrated in Figure 1.

From an SFST $\mathscr{A}$ a random path π can be drawn by starting at state q_0, drawing an outgoing edge with probability $w(e)$, and iterating this process at the next state $\text{tgt}(e)$. At any state q, terminate with probability $f(q)$.

Claim: If π is a random path through $\mathscr{A}$, then $(\text{istr}(\pi), \text{ostr}(\pi))$ is a random string pair distributed according to $[\![\mathscr{A}]\!]$. *Reason:* The morphism $g : E^* \to (\Gamma^* \times \Delta^*)$ defined for successful paths by $g(\pi) = (\text{istr}(\pi), \text{ostr}(\pi))$ has preimage $g^{-1}(u, v) = \text{Paths}(u, v)$, which is measurable by (1). By construction, π is an arbitrary path in this countable set of disjoint events and was drawn with probability $w(\pi)$.

In other words, we can sample random string pairs by sampling random paths. This allows us to gloss over the fact that an SFST $\mathscr{A}$ gives rise to both a probability space over paths and a probability space over string pairs. We write $X \sim \mathscr{A}$ to refer to a random string pair with distribution $[\![\mathscr{A}]\!]$.

It follows that sampling from SFSTs is unaffected by WFST optimizations which leave the behavior of an automaton intact. This includes weighted epsilon-removal, determinization, and minimization [10]; as well as disambiguation [11]. This means that $X \sim \mathscr{A}$ and $Y \sim g(\mathscr{A})$ are equal as random elements, for any combination of optimizations g. We can in effect sample Y by sampling paths from $\mathscr{A}$. As we will see below, this can be efficient in situations where determinization or disambiguation would lead to an exponential size increase.

Sampling from SFSTs can be effective even when operations would change their behavior. Isomorphisms such as FST

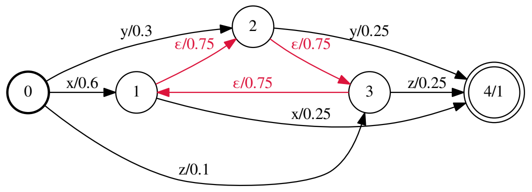

Figure 2: *Normalized SFST with epsilon-cycle highlighted.*

reversal or inversion are a particularly simple case: If we want to sample from the reverse of $\mathscr{A}$, we can either construct and then sample from WeightPush(Reverse($\mathscr{A}$)); or, equivalently, sample directly from $\mathscr{A}$ and then reverse the sampled paths/strings.

More generally, this strategy also works for certain morphisms of string pairs (such as morphisms of arcs that only change arc labels, including FST projection) as well as composition with unweighted functional FSTs. For example, sampling from $\text{Project}_1(\mathscr{A})$ is equivalent to drawing a random string pair (u_1, u_2) from $\mathscr{A}$ and returning (u_1, u_1).

1.2. Epsilon-cycle conflation

The strategy of sampling random string pairs by sampling random paths can be inefficient for SFSTs with cyclic epsilon graphs, especially in the presence of high-probability epsilon cycles. If the probability of the epsilon-loop at state 1 in Figure 1 is $1 - \delta$, then a random path will contain $1/\delta - 1$ repetitions of that edge on average, which can be problematic for small δ, i.e. loop probabilities close to 1. Sampling strings by sampling paths is not efficient in this situation, as the path sampling algorithm has to repeatedly expand the epsilon-loop despite the fact that it contributes no symbols to the string labeling.

One remedy is to apply the weighted epsilon-removal algorithm [10, pp. 233–37]. We illustrate this using an SFST with a nontrivial strongly-connected component (SCC) of its epsilon-graph, highlighted in Figure 2. The result of epsilon-removal is depicted in the left-hand side of Figure 3. Because epsilon-removal eliminated all epsilon-transitions, new outgoing transitions were added with appropriate weights for all outgoing transitions from any state in the original epsilon-SCC.

As we had seen above, in the context of sampling the presence of epsilon-transitions is not necessarily problematic. It would suffice to simply ensure that the epsilon-graph is acyclic. This can be achieved by a modification of the epsilon-removal algorithm, which we call epsilon-cycle conflation. Rather than removing all epsilon-transitions, this algorithm merely replaces every set of epsilon-paths through an epsilon-SCC with a single epsilon-transition. This is illustrated in the right-hand side of Figure 3; pseudocode appears in Figure 4.

The main property of the epsilon-cycle conflation algorithm is that it splits each state within an epsilon-SCC into two states, by adding a new state that is paired with an existing state. Transitions from within the SCC are removed (line 16); transitions from outside the SCC are redirected to reach the new state (line 18). Each SCC is replaced with a complete bipartite graph: for each newly split state, epsilon-transitions are added to all other original states within the SCC (for-loop starting on line 4). In other words, the bipartite graph that replaces the SCC encodes the all-pairs algebraic distance within the SCC. The algorithm references standard subroutines for computing the SCCs (line 2), for computing the single-source algebraic distance (line 6), and for trimming useless states (line 21), as well as standard WFST accessor functions for adding and deleting states and arcs. It also uses a simple arc-filter called EPSILON-

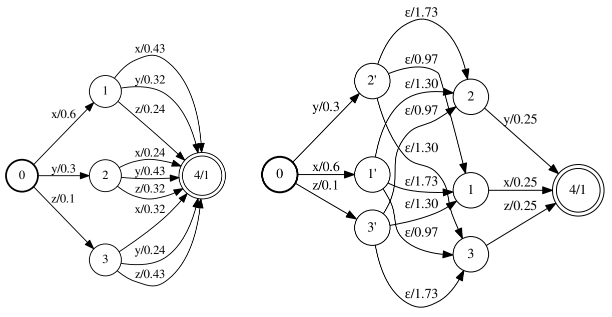

Figure 3: *SFSTs equivalent to Figure 2 after epsilon-removal (left) vs. epsilon-cycle conflation (right).*

```
 1: procedure CONFLATEEPSILONCYCLES($\mathscr{A}$)
 2:     $scc \leftarrow$ SCC($\mathscr{A}$)                    ▷ Map from state to SCC number
 3:     $split \leftarrow$ MAP()                                ▷ Empty map
 4:     for $(state, component)$ in $scc$ do
 5:         $filter \leftarrow$ EPSILONSCCARCFILTER($scc$, $component$)
 6:         $distance \leftarrow$ SHORTESTDISTANCE($state$, $filter$)
 7:         $s \leftarrow$ ADDSTATE($\mathscr{A}$)
 8:         for $(t, c)$ in $scc$ do
 9:             if $c = component$ then
10:                 ADDARC($\mathscr{A}$, $(s, \varepsilon, \varepsilon, distance[t], t)$)
11:                 $split[state] \leftarrow s$
12:     for $(state, component)$ in $scc$ do
13:         $filter \leftarrow$ EPSILONSCCARCFILTER($scc$, $component$)
14:         for $arc$ in ARCS($\mathscr{A}$, $state$) do
15:             if $filter(arc)$ then
16:                 DELETEARC($\mathscr{A}$, $arc$)
17:             else if tgt($arc$) in $split$ then
18:                 tgt($arc$) $\leftarrow split[$tgt($arc$)$]$
19:     if START($\mathscr{A}$) in $split$ then
20:         START($\mathscr{A}$) $\leftarrow split[$START($\mathscr{A}$)$]$
21:     CONNECT($\mathscr{A}$)
```

Figure 4: *Epsilon-cycle conflation algorithm.*

SCCARCFILTER (line 5 and 13), which is a predicate that returns true iff a given arc is an epsilon-transition within the specified SCC. It is assumed that the algebraic distance subroutine SHORTESTDISTANCE can take this filter as an argument, so as to restrict it to the specified SCC within the epsilon-graph.[1]

Unlike epsilon-removal, epsilon-cycle conflation only affects the states and arcs within each epsilon-SCC. It does not add or remove any non-epsilon transitions or epsilon-transitions not fully within epsilon-SCCs. If the sizes of the epsilon-SCCs are $S_1, \ldots, S_n$, epsilon-cycle conflation adds at most $\sum_{i=1}^{n} S_i$ additional states and no more than $\sum_{i=1}^{n} S_i^2$ additional transitions. A worst case arises if the entire FST is the cycle graph C_m with m states and m epsilon-arcs, in which case epsilon-cycle conflation will result in $2m$ states and m^2 arcs before trimming.

From here on we assume that any SFST we sample from is locally normalized and epsilon-cycle free. If it is not, we first apply epsilon-cycle conflation, followed by weight pushing.

2. CTC decoding by sampling

We now turn to an illustration of sampling from stochastic automata in the context of the decoding problem for Connectionist Temporal Classification (CTC) [13]. The CTC decoding problem arises because the sequences generated as CTC outputs do not correspond directly to the final label sequences. CTC output is a $T \times L$ dimensional matrix (w_{ij}) where each vector $w_{i \cdot}$ rep-

[1]This functionality is readily available in the OpenFst library [12], which forms the basis of our implementation.

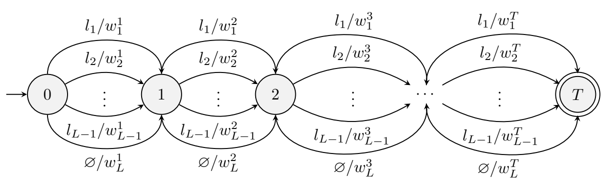

Figure 5: *CTC predictive distribution as a lattice WFSA $\mathscr{A}$.*

resents the probability distribution at time *i* over symbols from a finite alphabet of labels with cardinality $L-1$ augmented with a special blank symbol $\varnothing$.

A standard way (e.g. [14]) of conceptualizing CTC output is as an equivalent weighted finite state acceptor (WFSA). Figure 5 shows a WFSA view of a CTC lattice corresponding to a sequence of length T of symbols from $\{l_1, \ldots, l_{L-1}, \varnothing\}$. The cost of transitioning into state i having accepted symbol l_j is w_j^i.

From a path π through the CTC lattice we get to a final labeling sequence ℓ by first collapsing contiguous runs of repeated symbols and then removing blank symbols. For example, the path labeled with "aab∅b" gives rise to the labeling "abb". Following [13] we denote the path-to-labeling function as $\mathscr{B}$.

Crucially, the path-to-labeling function $\mathscr{B}$ can be computed by an unweighted functional FST, which we also refer to as $\mathscr{B}$. In order to collapse contiguous runs, the FST only needs to remember the last symbol seen, i.e. it has the structure of a bigram model over its input labels. This is illustrated in the top FST in Figure 6, where e.g. state 1 is reached whenever the input label ‘a’ is read in any state. On reaching state 1 from another state, ‘a’ is also output, but subsequent occurrences of ‘a’ are mapped to the empty string. A similar construction was used for the *Token WFST* in [14].

For a given SFST $\mathscr{A}$ over a label alphabet Γ and path-to-labeling transducer $\mathscr{B}$, the posterior predictive distribution over labeling sequences $\ell \in \Gamma^*$ corresponds to the output-projection of the WFST composition $\mathscr{A} \circ \mathscr{B}$:

$$p(\ell) = [\![\text{Project}_2(\mathscr{A} \circ \mathscr{B})]\!](\ell, \ell)\,. \tag{2}$$

In maximum a posteriori (MAP) decoding we are asked to find $\ell^\star = \text{argmax}_{\ell \in \Gamma^*}\, p(\ell)$. It is easy to define more complex decoding tasks over the labeling distribution p from (2).

The key point of this paper is that we can sample efficiently from the labeling distribution p for many choices of $\mathscr{A}$ and $\mathscr{B}$, including but not limited to CTC decoding, where exact MAP decoding via disambiguation or determinization is generally only tractable for unrealistically small WFSTs. When (2) involves only operations discussed in Section 1 above, we know that path-sampling is effective. All we have to do is sample paths from the SFST $\mathscr{A}$, pass them through the unweighted functional FST $\mathscr{B}$, and read labeling strings off the output tape.

This immediately leads us to a naive decoding strategy: Sample a large number of labeling strings and return the labeling which occurs most frequently in the sample, i.e. the sample mode. The advantage is that this is very easy to implement. On the other hand, the naive strategy often needs to sample a large number of paths, both to ensure that there are no unexplored modes and to distinguish labelings that are nearly equally likely.

The naive strategy can be improved by evaluating the probability $p(\ell)$ from (2) for a given labeling ℓ or set of labelings Y. We do this by expressing the preimage $\mathscr{B}^{-1}(Y) = \{x \mid \mathscr{B}(x) \in Y\}$ as a WFST $\mathscr{B} \circ Y$ (see the bottom FST in Figure 6) and measuring that preimage using $\mathscr{A}$. For singleton $Y = \{\ell\}$ the number of states and arcs of the preimage FST is linear in $|\ell|$ by

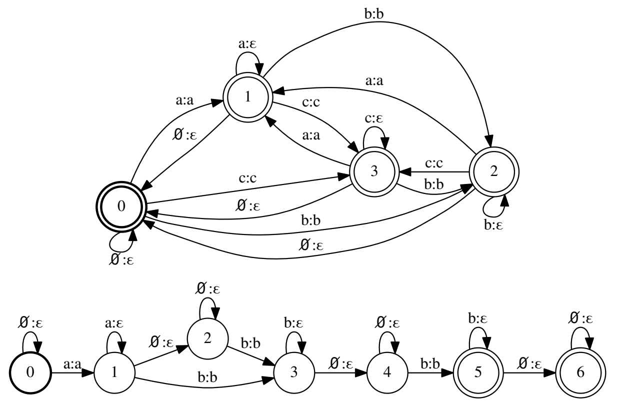

Figure 6: *Unweighted path-to-labeling FST $\mathscr{B}$ (top) for the symbol alphabet* {a, b, c} *plus blank symbol* $\varnothing$*; and composed FST* $\mathscr{B} \circ$ "abb" *(bottom) representing* $\mathscr{B}^{-1}$("abb").

construction. Evaluating $p(\ell)$ amounts to computing the grand total weight from (1) of $\mathscr{A} \circ (\mathscr{B} \circ \ell)$ using standard WFST operations. It is important to note that this efficiently sums over an implicitly represented WFST that compactly encodes an exponentially larger set of paths. This means that, while we could compute lower bounds for $p(\ell)$ via sampling and path probabilities alone, those bounds are so loose as to be of little practical value, due to the large number of paths for the same labeling.

The ability to evaluate (2) is useful in several ways. For example, if we know that $p(\ell) > 0.5$ for a particular labeling ℓ, then ℓ must necessarily be the mode of p. Generalizing from here gives us our sampling-based CTC decoding strategy (pseudocode appears in Figure 7): Repeatedly draw ℓ using path-based sampling (lines 12–13), compute $p(\ell)$ (line 19) and track the most probable labeling seen (lines 3–4 and 21–22) as well as the probability mass of all seen labelings (lines 10 and 20). We always include the labeling for the most likely path (line 2), a fast and useful heuristic already noted in [13]. If the probability of the most probable seen labeling exceeds the unseen probability mass (lines 5 and 23), then that labeling must be the global mode. For efficiency the decoding procedure takes parameters specifying a maximum number of random draws and a threshold θ for early stopping.

There are many ways to conceptualize when to stop sampling (line 25), most generally as an optimal stopping problem [15, 16], which subsumes sequential probability ratio tests. We want to stop sampling when we are reasonably certain that we have already seen the global mode.[2] Since in this strategy we can observe the probabilities of labelings, we could hypothesize that there is an unseen mode that occurs with probability P_0, which is a random variable that we assume follows a $\text{Beta}(1, n+1)$ distribution. The probability P_0 of the unseen mode would have to exceed the highest labeling probability $p^\star$ encountered so far, and cannot exceed the unseen probability mass $1-t$. We can stop sampling as soon as this becomes sufficiently unlikely, when $\Pr(p^\star \leq P_0 \leq 1-t) = (1-p^\star)^{n+1} - t^{n+1} < \theta$.

While the evaluation of labeling probabilities provides useful information that guides early stopping, it is often not necessary to compute $p(\ell)$ for every labeling string sampled randomly. Like the naive sampling strategy, our decoding algorithm keeps a count of labelings (lines 7–8 and 14–16). The

[2]The early stopping strategies are applicable because we can draw iid samples from the posterior predictive distribution. If instead we sampled correlated variates sequentially, there would be a substantial risk that e.g. a Markov chain sampler might get stuck near a local maximum.

```
1: function CtcDecode($\mathscr{A}$, $\mathscr{B}$, $max\_draws$, $\theta$)
2:     $\pi^\star \leftarrow$ ShortestPath($\mathscr{A}$)                ▷ most likely path
3:     $\ell^\star \leftarrow$ RmEpsilon(Project$_2$($\pi^\star \circ \mathscr{B}$))
4:     $p^\star \leftarrow$ ShortestDistance($\mathscr{A} \circ (\mathscr{B} \circ \ell^\star)$)
5:     if $p^\star > 0.5$ then
6:         return $\ell^\star$                              ▷ global mode found
7:     $C \leftarrow$ Map()                ▷ empty map of labelings to counts
8:     $C[\ell^\star] \leftarrow 1$
9:     $S \leftarrow \{\ell^\star\}$               ▷ set of labelings with known probability
10:    $t \leftarrow p^\star$                            ▷ total seen probability mass
11:    for $n \leftarrow 1 \ldots max\_draws$ do
12:        $\pi \leftarrow$ RandGen($\mathscr{A}$)                         ▷ random path
13:        $\ell \leftarrow$ RmEpsilon(Project$_2$($\pi \circ \mathscr{B}$))
14:        if $\ell \notin C$ then
15:            $C[\ell] \leftarrow 0$
16:        $c \leftarrow C[\ell] \leftarrow C[\ell] + 1$
17:        if $\ell \notin S$ and ComputeProbability($c$, $n$, $p^\star$, $t$, $\theta$) then
18:            $S \leftarrow S \cup \{\ell\}$
19:            $p \leftarrow$ ShortestDistance($\mathscr{A} \circ (\mathscr{B} \circ \ell)$)
20:            $t \leftarrow t + p$
21:            if $p > p^\star$ then
22:                $\ell^\star, p^\star \leftarrow \ell, p$
23:            if $p^\star > 1 - t$ then
24:                break                              ▷ global mode found
25:        if $\Pr(p^\star \leq P_0 \leq 1 - t) < \theta$ then   ▷ for $P_0 \sim \text{Beta}(1, n+1)$
26:            break                       ▷ approximate early stopping
27:    return $\ell^\star$
```

Figure 7: *Sampling-based CTC decoding algorithm.*

count c (line 16) of the current labeling ℓ, together with other information, is passed to a predicate ComputeProbability indicating whether $p(\ell)$ should be evaluated during a given iteration (the *probability computation strategy*), which can be implemented in a variety of ways. If it is always true, probabilities are computed for every distinct labeling, which can improve early stopping. If it is always false and if the most frequent labeling in C is returned, this becomes the naive sampling algorithm. ComputeProbability could also be implemented (similar to the negation of the early stopping criterion) as $\Pr(p^\star \leq P_c \leq 1-t) \geq \theta$ where P_c is a random variable with a $\text{Beta}(c+1, n-c+2)$ distribution, so we compute $p(\ell)$ only when this value is likely to exceed $p^\star$. Finally, the simple criterion $c > 1$ works well in our experience: only evaluate the probability of a labeling when seeing it for the second time.

The decoding algorithm can thus be fine-tuned for many practical considerations, and many additional variations are possible, such as sampling in batches and only occasionally testing for early stopping. In the following section we compare different decoding strategies, including the decoding algorithm from Figure 7 and the naive sampling algorithm.

3. Evaluation and discussion

Our experiments focus on continuous phoneme recognition, which is a sequence-to-sequence task where the input sequence corresponds to the acoustic features and the output is a sequence of categorical labels representing phonemes from a finite inventory. CTC-based approaches have been successfully applied to this task [17, 18, 19, 20]. We trained a CTC phoneme recognizer for Argentinian Spanish [21] based on the architecture described in [22]. We computed CTC output matrices on a test set of 90 unseen utterances and turned those into CTC lattices [23].

We compared different decoding strategies in terms of their ability to find the global mode of the posterior predictive labeling distribution. We first ran an exhaustive search to locate the global mode. We then compared different decoding strategies in terms of how often they were able to locate the global mode. The results are summarized in Table 1. The first two rows refer to the CTC beam search decoding algorithm of TensorFlow [24], which we ran with its default beam-width of 100 and then again with a much larger beam-width of 2000, which did not affect the results of beam search.

Table 1: *Comparison of decoding strategies in terms of success at finding the global mode and mean number of paths sampled.*

	Decoding strategy, probability computation strategy	Mode found	Avg. paths sampled	Avg. probs. computed
1	BeamSearch(100), n/a	83%		
2	BeamSearch(2000), n/a	83%		
3	NaiveSampling(600, $\theta = 0$), never	82%	600	0
4	NaiveSampling(6000, $\theta = 0$), never	94%	6000	0
5	CtcDecode(0, $\theta = 0$), never	77%	0	0
6	CtcDecode(100, $\theta = 0.01$), always	99%	36	27
7	CtcDecode(600, $\theta = 0.01$), always	100%	53	40
8	CtcDecode(600, $\theta = 0.01$), if $c > 1$	100%	53	7

The third and fourth row of Table 1 show results for the naive sampling strategy, which simply returns the mode of a large sample. Because no early stopping is employed, the average number of paths sampled per utterance is identical to the *max_draws* parameter of Figure 7. In the naive sampling strategy the number of probability computations is zero by design. This strategy can match or exceed beam-search in its ability to locate the global mode, but this comes at a substantial cost, as a large number of paths needs to be sampled. We experimented with early stopping for naive sampling before deciding against it: while early stopping can ensure that there are no unseen modes, a reduction in sample size made it much harder to correctly identify the global mode within the sample.

The fifth row of Table 1 is the best-path heuristic mentioned in [13]. It only considers the labeling of the most likely path $\pi^\star$ and does not sample any additional paths. While it performs worst in this comparison, it is a useful heuristic due to speed.

The last three rows of Table 1 are nontrivial instances of our decoding algorithm [25]. We fixed a confidence threshold of $\theta = 0.01$. Larger values result in fewer overall computations, at the expense of possibly not finding the global mode. Smaller values increase the overall computations. We then chose a maximum number of 600 draws, which always leads to early stopping: the algorithm from Figure 7 either finds the global mode (lines 6 and 24) or stops early with confidence θ (line 26). When running with a sample limit of 600 paths, our algorithm always found the global mode in our experiments. Even with a much smaller limit of at most 100 random draws, our algorithm still did well. Finally, as the last row shows, the average number of probability evaluations can be greatly reduced by computing labeling probabilities only when a labeling is encountered more than once. The last three strategies shown here all outperform beam search.

4. Conclusion

We have discussed sufficient conditions under which sampling random paths from transformations of stochastic automata yields random strings from the transformed automata. We have shown that this can be used to sample labeling sequences from CTC posterior lattices. In a comparison on a connected phoneme recognition problem, sampling-based decoding informed by posterior probabilities was able to locate posterior modes reliably with minimal search effort.

5. References

[1] M. Mohri, F. Pereira, and M. Riley, "Weighted Finite-State Transducers in Speech Recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[2] S. M. Siniscalchi, T. Svendsen, and C.-H. Lee, "A Bottom-Up Stepwise Knowledge-Integration Approach to Large Vocabulary Continuous Speech Recognition Using Weighted Finite State Machines," in *Proc. of the 12th Annual Conference of the International Speech Communication Association (Interspeech-2011)*, Florence, Italy, 2011, pp. 901–904.

[3] C. Allauzen, M. Mohri, and M. Riley, "Statistical Modeling for Unit Selection in Speech Synthesis," in *Proc. of the 42nd Annual Meeting on Association for Computational Linguistics (ACL)*, 2004, pp. 55–62.

[4] I. Bulyko, "Flexible Speech Synthesis Using Weighted Finite-State Transducers," Ph.D. dissertation, University of Washington, 2002.

[5] F. Casacuberta and E. Vidal, "Machine Translation with Inferred Stochastic Finite-State Transducers," *Computational Linguistics*, vol. 30, no. 2, pp. 205–225, 2004.

[6] K. Wu, C. Allauzen, K. Hall, M. Riley, and B. Roark, "Encoding linear models as weighted finite-state transducers," in *Proc. of 15th Annual Conference of the International Speech Communication Association (Interspeech-2014)*, Singapore, 2014, pp. 1258–1262.

[7] E. Tzoukermann and D. R. Radev, "Use of Weighted Finite State Transducers in Part of Speech Tagging," *arXiv preprint cmp-lg/9710001*, 1997.

[8] H. Sak, T. Güngör, and M. Saraçlar, "A Stochastic Finite-State Morphological Parser for Turkish," in *Proc. of the ACL-IJCNLP 2009 Conference*, Singapore, 2009, pp. 273–276.

[9] L. Hellsten, B. Roark, P. Goyal, C. Allauzen, F. Beaufays, T. Ouyang, M. Riley, and D. Rybach, "Transliterated Mobile Keyboard Input via Weighted Finite-State Transducers," in *Proc. of the 13th International Conference on Finite State Methods and Natural Language Processing (FSMNLP 2017)*, Umeå, Sweden, 2017, pp. 10–19.

[10] M. Mohri, "Weighted automata algorithms," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, Eds. Springer, 2009, pp. 213–254.

[11] M. Mohri and M. D. Riley, "A Disambiguation Algorithm for Weighted Automata," *Theoretical Computer Science*, vol. 679, pp. 53–68, 2017.

[12] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A General and Efficient Weighted Finite-State Transducer Library," in *CIAA 2007: Implementation and Application of Automata*, 2007, pp. 11–23.

[13] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proc. 23rd International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, Jun. 2006, pp. 369–376.

[14] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Automatic Speech Recognition and Understanding Workshop (ASRU)*, Scottsdale, Arizona, Dec. 2015.

[15] F. T. Bruss, "Sum the odds to one and stop," *Annals of Probability*, vol. 28, no. 3, pp. 1384–91, 2000.

[16] T. S. Ferguson, "Optimal stopping and applications," 2008, http://www.math.ucla.edu/~tom/Stopping/.

[17] S. Fernández, A. Graves, and J. Schmidhuber, "Phoneme recognition in TIMIT with BLSTM-CTC," *arXiv preprint arXiv:0804.3269*, 2008.

[18] A. Graves, A. R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Canada, May 2013, pp. 6645–6649.

[19] H. Sak, F. de Chaumont Quitry, T. Sainath, K. Rao *et al.*, "Acoustic modelling with CD-CTC-SMBR LSTM RNNs," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Scottsdale, Arizona, Dec. 2015, pp. 604–609.

[20] G. Pundak and T. N. Sainath, "Lower Frame Rate Neural Network Acoustic Models," in *Interspeech 2016*, San Francisco, USA, Sep. 2016, pp. 22–26.

[21] "Crowdsourced high-quality Argentinian Spanish speech data set," OpenSLR, resource SLR61, 2019. [Online]. Available: http://openslr.org/61

[22] O. Adams, T. Cohn, G. Neubig, H. Cruz, S. Bird, and A. Michaud, "Evaluating phonemic transcription of low-resource tonal languages for language documentation," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Japan, 2018, pp. 3356–3365.

[23] "Argentinian Spanish CTC phoneme lattices," 2019. [Online]. Available: https://github.com/google/language-resources/tree/master/es

[24] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016, pp. 265–283.

[25] "Sampling-based mode search," 2019. [Online]. Available: https://github.com/mjansche/ctc_sampling