A Dynamic Programming Algorithm for Computing N-gram Posteriors from Lattices

Doğan Can
Department of Computer Science
University of Southern California
Los Angeles, CA, USA
dogancan@usc.edu

Shrikanth S. Narayanan
Department of Electrical Engineering
University of Southern California
Los Angeles, CA, USA
shri@sipi.usc.edu

Abstract

Efficient computation of n-gram posterior probabilities from lattices has applications in lattice-based minimum Bayes-risk decoding in statistical machine translation and the estimation of expected document frequencies from spoken corpora. In this paper, we present an algorithm for computing the posterior probabilities of all n-grams in a lattice and constructing a minimal deterministic weighted finite-state automaton associating each n-gram with its posterior for efficient storage and retrieval. Our algorithm builds upon the best known algorithm in literature for computing n-gram posteriors from lattices and leverages the following observations to significantly improve the time and space requirements: i) the n-grams for which the posteriors will be computed typically comprises all n-grams in the lattice up to a certain length, ii) posterior is equivalent to expected count for an n-gram that do not repeat on any path, iii) there are efficient algorithms for computing n-gram expected counts from lattices. We present experimental results comparing our algorithm with the best known algorithm in literature as well as a baseline algorithm based on weighted finite-state automata operations.

1 Introduction

Many complex speech and natural language processing (NLP) pipelines such as Automatic Speech Recognition (ASR) and Statistical Machine Translation (SMT) systems store alternative hypotheses produced at various stages of processing as weighted acyclic automata, also known as lattices. Each lattice stores a large number of hypotheses along with the raw system scores assigned to them. While single-best hypothesis is typically what is desired at the end of the processing, it is often beneficial to consider a large number of weighted hypotheses at earlier stages of the pipeline to hedge against errors introduced by various subcomponents. Standard ASR and SMT techniques like discriminative training, rescoring with complex models and Minimum Bayes-Risk (MBR) decoding rely on lattices to represent intermediate system hypotheses that will be further processed to improve models or system output. For instance, lattice based MBR decoding has been shown to give moderate yet consistent gains in performance over conventional MAP decoding in a number of speech and NLP applications including ASR (Goel and Byrne, 2000) and SMT (Tromble et al., 2008; Blackwood et al., 2010; de Gispert et al., 2013).

Most lattice-based techniques employed by speech and NLP systems make use of posterior quantities computed from probabilistic lattices. In this paper, we are interested in two such posterior quantities: i) n-gram expected count, the expected number of occurrences of a particular n-gram in a lattice, and ii) n-gram posterior probability, the total probability of accepting paths that include a particular n-gram. Expected counts have applications in the estimation of language model statistics from probabilistic input such as ASR lattices (Allauzen et al., 2003) and the estimation term frequencies from spoken corpora while posterior probabilities come up in MBR decoding of SMT lattices (Tromble et al., 2008), relevance ranking of spoken utterances and the estimation of document frequencies from spoken corpora (Karakos et al., 2011; Can and Narayanan, 2013).

The expected count $c(x|A)$ of n-gram x given lattice A is defined as

$$c(x|A) = \sum_{y \in \Sigma^*} \#_y(x) p(y|A) \qquad (1)$$

where $\#_y(x)$ is the number of occurrences of n-

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2388–2397,
Lisbon, Portugal, 17-21 September 2015.

gram x in hypothesis y and $p(y|A)$ is the posterior probability of hypothesis y given lattice A. Similarly, the posterior probability $p(x|A)$ of n-gram x given lattice A is defined as

$$p(x|A) = \sum_{y \in \Sigma^*} 1_y(x) p(y|A) \quad (2)$$

where $1_y(x)$ is an indicator function taking the value 1 when hypothesis y includes n-gram x and 0 otherwise. While it is straightforward to compute these posterior quantities from weighted n-best lists by examining each hypothesis separately and keeping a separate accumulator for each observed n-gram type, it is infeasible to do the same with lattices due to the sheer number of hypotheses stored. There are efficient algorithms in literature (Allauzen et al., 2003; Allauzen et al., 2004) for computing n-gram expected counts from weighted automata that rely on weighted finite state transducer operations to reduce the computation to a sum over n-gram occurrences eliminating the need for an explicit sum over accepting paths. The rather innocent looking difference between Equations 1 and 2, $\#_y(x)$ vs. $1_y(x)$, makes it hard to develop similar algorithms for computing n-gram posteriors from weighted automata since the summation of probabilities has to be carried out over paths rather than n-gram occurrences (Blackwood et al., 2010; de Gispert et al., 2013).

The problem of computing n-gram posteriors from lattices has been addressed by a number of recent works (Tromble et al., 2008; Allauzen et al., 2010; Blackwood et al., 2010; de Gispert et al., 2013) in the context of lattice-based MBR for SMT. In these works, it has been reported that the time required for lattice MBR decoding is dominated by the time required for computing n-gram posteriors. Our interest in computing n-gram posteriors from lattices stems from its potential applications in spoken content retrieval (Chelba et al., 2008; Karakos et al., 2011; Can and Narayanan, 2013). Computation of document frequency statistics from spoken corpora relies on estimating n-gram posteriors from ASR lattices. In this context, a spoken document is simply a collection of ASR lattices. The n-grams of interest can be word, syllable, morph or phoneme sequences. Unlike in the case of lattice-based MBR for SMT where the n-grams of interest are relatively short – typically up to 4-grams –, the n-grams we are interested in are in many instances relatively long sequences of subword units.

In this paper, we present an efficient algorithm for computing the posterior probabilities of all n-grams in a lattice and constructing a minimal deterministic weighted finite-state automaton associating each n-gram with its posterior for efficient storage and retrieval. Our n-gram posterior computation algorithm builds upon the custom forward procedure described in (de Gispert et al., 2013) and introduces a number of refinements to significantly improve the time and space requirements:

- The custom forward procedure described in (de Gispert et al., 2013) computes unigram posteriors from an input lattice. Higher order n-gram posteriors are computed by first transducing the input lattice to an n-gram lattice using an order mapping transducer and then running the custom forward procedure on this higher order lattice. We reformulate the custom forward procedure as a dynamic programming algorithm that computes posteriors for successively longer n-grams and reuses the forward scores computed for the previous order. This reformulation subsumes the transduction of input lattices to n-gram lattices and obviates the need for constructing and applying order mapping transducers.

- Comparing Eq. 1 with Eq. 2, we can observe that posterior probability and expected count are equivalent for an n-gram that do not repeat on any path of the input lattice. The key idea behind our algorithm is to limit the costly posterior computation to only those n-grams that can potentially repeat on some path of the input lattice. We keep track of repeating n-grams of order n and use a simple impossibility argument to significantly reduce the number of n-grams of order $n+1$ for which posterior computation will be performed. The posteriors for the remaining n-grams are replaced with expected counts. This filtering of n-grams introduces a slight bookkeeping overhead but in return dramatically reduces the runtime and memory requirements for long n-grams.

- We store the posteriors for n-grams that can potentially repeat on some path of the input lattice in a weighted prefix tree that we construct on the fly. Once that is done, we com-

Table 1: Common semirings.

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

$a \oplus_{\log} b = -\log(e^{-a} + e^{-b})$

pute the expected counts for all n-grams in the input lattice and represent them as a minimal deterministic weighted finite-state automaton, known as a factor automaton (Allauzen et al., 2004; Mohri et al., 2007), using the approach described in (Allauzen et al., 2004). Finally we use general weighted automata algorithms to merge the weighted factor automaton representing expected counts with the weighted prefix tree representing posteriors to obtain a weighted factor automaton representing posteriors that can be used for efficient storage and retrieval.

2 Preliminaries

This section introduces the definitions and notation related to weighted finite state automata and transducers (Mohri, 2009).

2.1 Semirings

Definition 1 *A semiring is a 5-tuple* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *where* $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid,* $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid,* $\otimes$ *distributes over* $\oplus$ *and* $\overline{0}$ *is an annihilator for* $\otimes$.

Table 1 lists common semirings. In speech and language processing, two semirings are of particular importance. The *log semiring* is isomorphic to the probability semiring via the negative-log morphism and can be used to combine probabilities in the log domain. The *tropical semiring*, provides the algebraic structure necessary for shortest-path algorithms and can be derived from the log semiring using the Viterbi approximation.

2.2 Weighted Finite-State Automata

Definition 2 *A weighted finite-state automaton (WFSA)* A *over a semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is a 7-tuple* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ *where:* Σ *is the finite input alphabet;* Q *is a finite set of states;* $I, F \subseteq Q$ *are respectively the set of initial and final states;* $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ *is a finite set of arcs;* $\lambda : I \to \mathbb{K}$, $\rho : F \to \mathbb{K}$ *are respectively the initial and final weight functions.*

Given an arc $e \in E$, we denote by $i[e]$ its input label, $w[e]$ its weight, $s[e]$ its source or origin state and $t[e]$ its target or destination state. A path $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive arcs satisfying $t[e_{i-1}] = s[e_i]$, $i = 2, \ldots, k$. We extend t and s to paths by setting $t[\pi] = s[e_k]$ and $s[\pi] = t[e_1]$. The labeling and the weight functions can also be extended to paths by defining $i[\pi] = i[e_1] \ldots i[e_k]$ and $w[\pi] = w[e_1] \otimes \ldots \otimes w[e_k]$. We denote by $\Pi(q, q')$ the set of paths from q to q' and by $\Pi(q, x, q')$ the set of paths from q to q' with input string $x \in \Sigma^*$. These definitions can also be extended to subsets $S, S' \subseteq Q$, e.g.

$$\Pi(S, x, S') = \bigcup_{q \in S, q' \in S'} \Pi(q, x, q').$$

An *accepting* path in an automaton A is a path in $\Pi(I, F)$. A string x is accepted by A if there exists an accepting path π labeled with x. A is *deterministic* if it has at most one initial state and at any state no two outgoing transitions share the same input label. The weight associated by an automaton A to a string $x \in \Sigma^*$ is given by

$$[\![A]\!](x) = \bigoplus_{\pi \in \Pi(I, x, F)} \lambda(s[\pi]) \otimes w[\pi] \otimes \rho(t[\pi])$$

and $[\![A]\!](x) \triangleq \overline{0}$ when $\Pi(I, x, F) = \emptyset$.

A weighted automaton A defined over the probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$ is said to be *probabilistic* if for any state $q \in Q$, the sum of the weights of all cycles at q, $\oplus_{\pi \in \Pi(q,q)} w[\pi]$, is well-defined and in $\mathbb{R}_+$ and $\sum_{x \in \Sigma^*} [\![A]\!](x) = 1$.

2.3 N-gram Mapping Transducer

We denote by Φ_n the n-gram mapping transducer (Blackwood et al., 2010; de Gispert et al., 2013)

of order n. This transducer maps label sequences to n-gram sequences of order n. Φ_n is similar in form to the weighted finite-state transducer representation of a backoff n-gram language model (Allauzen et al., 2003). We denote by A_n the n-gram lattice of order n obtained by composing lattice A with Φ_n, projecting the resulting transducer onto its output labels, i.e. n-grams, to obtain an automaton, removing ε-transitions, determinizing and minimizing (Mohri, 2009). A_n is a compact lattice of n-gram sequences of order n consistent with the labels and scores of lattice A. A_n typically has more states than A due to the association of distinct n-gram histories with states.

2.4 Factor Automata

Definition 3 *Given two strings $x, y \in \Sigma^*$, x is a factor (substring) of y if $y = uxv$ for some $u, v \in \Sigma^*$. More generally, x is a factor of a language $L \subseteq \Sigma^*$ if x is a factor of some string $y \in L$. The factor automaton $S(y)$ of a string y is the minimal deterministic finite-state automaton recognizing exactly the set of factors of y. The factor automaton $S(A)$ of an automaton A is the minimal deterministic finite-state automaton recognizing exactly the set of factors of A, that is the set of factors of the strings accepted by A.*

Factor automaton (Mohri et al., 2007) is an efficient and compact data structure for representing a full index of a set of strings, i.e. an automaton. It can be used to determine if a string x is a factor in time linear in its length $O(|x|)$. By associating a weight with each factor, we can generalize the factor automaton structure to weighted automata and use it for efficient storage and retrieval of n-gram posteriors and expected counts.

3 Computation of N-gram Posteriors

In this section we present an efficient algorithm based on the n-gram posterior computation algorithm described in (de Gispert et al., 2013) for computing the posterior probabilities of all n-grams in a lattice and constructing a weighted factor automaton for efficient storage and retrieval of these posteriors. We assume that the input lattice is an ε-free acyclic probabilistic automaton. If that is not the case, we can use general weighted automata ε-removal and weight-pushing algorithms (Mohri, 2009) to preprocess the input automaton.

Algorithm 1 reproduces the original algorithm of (de Gispert et al., 2013) in our notation. Each iteration of the outermost loop starting at line 1 computes posterior probabilities of all unigrams in the n-gram lattice $A_n = (\Sigma_n, Q_n, I_n, F_n, E_n, \lambda_n, \rho_n)$, or equivalently all n-grams of order n in the lattice A. The inner loop starting at line 6 is essentially a custom forward procedure computing not only the standard forward probabilities $\alpha[q]$, the marginal probability of paths that lead to state q,

$$\alpha[q] = \bigoplus_{\pi \in \Pi(I,q)} \lambda(s[\pi]) \otimes w[\pi] \tag{3}$$

$$= \bigoplus_{\substack{e \in E \\ t[e]=q}} \alpha[s[e]] \otimes w[e] \tag{4}$$

but also the label specific forward probabilities $\tilde{\alpha}[q][x]$, the marginal probability of paths that lead to state q and include label x.

$$\tilde{\alpha}[q][x] = \bigoplus_{\substack{\pi \in \Pi(I,q) \\ \exists\, u,v \in \Sigma^*:\ i[\pi]=uxv}} \lambda(s[\pi]) \otimes w[\pi] \tag{5}$$

$$= \bigoplus_{\substack{e \in E \\ t[e]=q \\ i[e]=x}} \alpha[s[e]] \otimes w[e]$$

$$\oplus \bigoplus_{\substack{e \in E \\ t[e]=q \\ i[e]\neq x}} \tilde{\alpha}[s[e]][x] \otimes w[e] \tag{6}$$

Just like in the case of the standard forward algorithm, visiting states in topological order ensures that forward probabilities associated with a state has already been computed when that state is visited. At each state s, the algorithm examines each arc $e = (s, x, w, q)$ and updates the forward probabilities for state q in accordance with the recursions in Equations 4 and 6 by propagating the forward probabilities computed for s (lines 8-12). The conditional on line 11 ensures that the label specific forward probability $\tilde{\alpha}[s][y]$ is propagated to state q only if label y is different from label x, the label on the current arc. In other words, if a label y repeats on some path π leading to state q, then π contributes to $\tilde{\alpha}[q][y]$ only once. This is exactly what is required by the indicator function in Equation 2 when computing unigram posteriors. Whenever a final state is processed, the posterior probability accumulator for each label observed on paths reaching that state is updated by multiplying the label specific forward probability and the final weight associated with that state

Algorithm 1 Compute N-gram Posteriors

```
1  for n ← 1, ..., N do
2      A_n ← Min(Det(RmEps(ProjOut(A ∘ Φ_n))))
3      α[q] ← λ_n(q), ∀ state q ∈ Q_n
4      α̃[q][x] ← 0̄, ∀ state q ∈ Q_n, ∀ label x ∈ Σ_n
5      p(x|A) ← 0̄, ∀ label x ∈ Σ_n
6      for each state s ∈ Q_n do                ▷ In topological order
7          for each arc (s, x, w, q) ∈ E_n do
8              α[q] ← α[q] ⊕ α[s] ⊗ w
9              α̃[q][x] ← α̃[q][x] ⊕ α[s] ⊗ w
10             for each label y ∈ α̃[s] do
11                 if y ≠ x then
12                     α̃[q][y] ← α̃[q][y] ⊕ α̃[s][y] ⊗ w
13         if s ∈ F_n then
14             for each label x ∈ α̃[s] do
15                 p(x|A) ← p(x|A) ⊕ α̃[s][x] ⊗ ρ_n(s)
16 P ← Min(ConstructPrefixTree(p))
```

and adding the resulting value to the accumulator (lines 13-15). It should be noted that this algorithm is a form of marginalization (de Gispert et al., 2013), rather than a counting procedure, due to the conditional on line 11. If that conditional were to be removed, this algorithm would compute n-gram expected counts instead of posterior probabilities.

The key idea behind our algorithm is to restrict the computation of posteriors to only those n-grams that may potentially repeat on some path of the input lattice and exploit the equivalence of expected counts and posterior probabilities for the remaining n-grams. It is possible to extend Algorithm 1 to implement this restriction by keeping track of repeating n-grams of order n and replacing the output labels of appropriate arcs in Φ_{n+1} with ε labels. Alternatively we can reformulate Algorithm 1 as in Algorithm 2. In this formulation we compute n-gram posteriors directly on the input lattice A without constructing the n-gram lattice A_n. We explicitly associate states in the original lattice with distinct n-gram histories which is implicitly done in Algorithm 1 by constructing the n-gram lattice A_n. This explicit association lets us reuse forward probabilities computed at order n while computing the forward probabilities at order $n+1$. Further, we can directly restrict the n-grams for which posterior computation will be performed.

In Algorithm 2, $\acute{\alpha}[n][q][h]$ represents the history specific forward probability of state q, the marginal probability of paths that lead to state q and include length n string h as a suffix.

$$\acute{\alpha}[n][q][h] = \bigoplus_{\substack{\pi \in \Pi(I,q) \\ \exists\, z \in \Sigma^* \colon\, i[\pi] = zh}} \lambda(s[\pi]) \otimes w[\pi] \tag{7}$$

$$= \bigoplus_{\substack{e \in E \\ t[e] = q \\ g \in \acute{\alpha}[n-1][s[e]] \\ gi[e] = h}} \acute{\alpha}[n-1][s[e]][g] \otimes w[e] \tag{8}$$

$\acute{\alpha}[n][q][h]$ is the analogue of $\alpha[q]$ in Algorithm 1. It splits the forward probability of state q (Equation 3), among length n suffixes (or histories) of paths that lead to state q. We can interpret $\acute{\alpha}[n][q][h]$ as the forward probability of state (q, h) in the n-gram lattice A_{n+1}. Here $(q, h) \in Q_{n+1}$ denotes the unique state corresponding to state q in the original lattice A and state h in the mapping transducer Φ_{n+1}. $\hat{\alpha}[q][h][x]$ represents the history and n-gram specific forward probability of state q, the marginal probability of paths that lead to state q, include length $n-1$ string h as a suffix and

Algorithm 2 Compute N-gram Posteriors (Reformulation)

1 $R[0] \leftarrow \{\varepsilon\}$
2 $\acute{\alpha}[0][q][\varepsilon] \leftarrow \alpha[q],\ \forall$ state $q \in Q$
3 **for** $n \leftarrow 1, \ldots, N$ **do**
4 $\quad R[n] \leftarrow \emptyset$
5 $\quad \acute{\alpha}[n][q][x] \leftarrow \overline{0},\ \forall$ state $q \in Q,\ \forall$ ngram $x \in \Sigma^n$
6 $\quad \hat{\alpha}[q][h][x] \leftarrow \overline{0},\ \forall$ state $q \in Q,\ \forall$ history $h \in \Sigma^{n-1},\ \forall$ ngram $x \in \Sigma^n$
7 $\quad p(x|A) \leftarrow \overline{0},\ \forall$ ngram $x \in \Sigma^n$
8 $\quad$ **for each** state $s \in Q$ **do** ▷ In topological order
9 $\quad\quad$ **for each** history $g \in \acute{\alpha}[n-1][s]$ **where** $g \in R[n-1]$ **do**
10 $\quad\quad\quad$ **for each** arc $(s, i, w, q) \in E$ **do**
11 $\quad\quad\quad\quad x \leftarrow gi$ ▷ Concatenate history and label
12 $\quad\quad\quad\quad h \leftarrow x[1:n]$ ▷ Drop first label
13 $\quad\quad\quad\quad$ **if** $h \in R[n-1]$ **then**
14 $\quad\quad\quad\quad\quad \acute{\alpha}[n][q][x] \leftarrow \acute{\alpha}[n][q][x] \oplus \acute{\alpha}[n-1][s][g] \otimes w$
15 $\quad\quad\quad\quad\quad \hat{\alpha}[q][h][x] \leftarrow \hat{\alpha}[q][h][x] \oplus \acute{\alpha}[n-1][s][g] \otimes w$
16 $\quad\quad\quad\quad\quad$ **for each** ngram $y \in \hat{\alpha}[s][g]$ **do**
17 $\quad\quad\quad\quad\quad\quad$ **if** $y \neq x$ **then**
18 $\quad\quad\quad\quad\quad\quad\quad \hat{\alpha}[q][h][y] \leftarrow \hat{\alpha}[q][h][y] \oplus \hat{\alpha}[s][g][y] \otimes w$
19 $\quad\quad\quad\quad\quad\quad$ **else**
20 $\quad\quad\quad\quad\quad\quad\quad R[n] \leftarrow R[n] \cup \{y\}$
21 $\quad\quad$ **if** $s \in F$ **then**
22 $\quad\quad\quad$ **for each** history $g \in \hat{\alpha}[s]$ **do**
23 $\quad\quad\quad\quad$ **for each** ngram $x \in \hat{\alpha}[s][g]$ **do**
24 $\quad\quad\quad\quad\quad p(x|A) \leftarrow p(x|A) \oplus \hat{\alpha}[s][g][x] \otimes \rho(s)$
25 $P' \leftarrow \text{ConstructPrefixTree}(p)$
26 $C \leftarrow \text{ComputeExpectedCounts}(A, N)$
27 $P \leftarrow \text{Min}(\text{Det}(\text{RmEps}((C - \text{RmWeight}(P')) \oplus P')))$

include n-gram x as a substring.

$$\hat{\alpha}[q][h][x] = \bigoplus_{\substack{\pi \in \Pi(I,q) \\ \exists\, z \in \Sigma^*:\ i[\pi] = zh \\ \exists\, u,v \in \Sigma^*:\ i[\pi] = uxv}} \lambda(s[\pi]) \otimes w[\pi] \tag{9}$$

$$= \bigoplus_{\substack{e \in E \\ t[e] = q \\ g \in \acute{\alpha}[|h|][s[e]] \\ gi[e] = x}} \acute{\alpha}[|h|][s[e]][g] \otimes w[e] \;\oplus \bigoplus_{\substack{e \in E \\ t[e] = q \\ g \in \hat{\alpha}[s[e]] \\ gi[e] \neq x}} \hat{\alpha}[s[e]][g][x] \otimes w[e] \tag{10}$$

$\hat{\alpha}[q][h][x]$ is the analogue of $\tilde{\alpha}[q][x]$ in Algorithm 1. $R[n]$ represents the set of n-grams of order n that repeat on some path of A. We start by defining $R[0] \triangleq \{\varepsilon\}$, i.e. the only repeating n-gram of order 0 is the empty string ε, and computing $\acute{\alpha}[0][q][\varepsilon] \equiv \alpha[q]$ using the standard forward algorithm. Each iteration of the outermost loop starting at line 3 computes posterior probabilities of all n-grams of order n directly on the lattice A. At iteration n, we visit the states in topological order and examine each length $n-1$ history g associated with s, the state we are in. For each history g, we go over the set of arcs leaving state s, construct the current n-gram x by concatenating g with the current arc label i (line 11), construct the length $n-1$ history h of the target state q (line 12), and update the forward probabilities for the target state history pair (q, h) in accordance with the recursions in Equations 8 and 10 by propagating the forward probabilities computed for the state history pair (s, g) (lines 14-18). Whenever a final state is processed, the posterior probability accumulator for

each n-gram of order n observed on paths reaching that state is updated by multiplying the n-gram specific forward probability and the final weight associated with that state and adding the resulting value to the accumulator (lines 21-24).

We track repeating n-grams of order n to restrict the costly posterior computation operation to only those n-grams of order $n+1$ that can potentially repeat on some path of the input lattice. The conditional on line 17 checks if any of the n-grams observed on paths reaching state history pair (s, g) is the same as the current n-gram x, and if so adds it to the set of repeating n-grams. At each iteration n, we check if the current length $n-1$ history g of the state we are in is in $R[n-1]$, the set of repeating n-grams of order $n-1$ (line 9). If it is not, then no n-gram $x = gi$ can repeat on some path of A since that would require g to repeat as well. If g is in $R[n-1]$, then for each arc $e = (s, i, w, q)$ we check if the length $n-1$ history $h = g[1:n-1]i$ of the next state q is in $R[n-1]$ (line 13). If it is not, then the n-gram $x = g[0]h$ can not repeat either.

We keep the posteriors $p(x|A)$ for n-grams that can potentially repeat on some path of the input lattice in a deterministic WFSA P' that we construct on the fly. P' is a prefix tree where each path π corresponds to an n-gram posterior, i.e. $i[\pi] = x \implies w[\pi] = \rho(t[\pi]) = p(x|A)$. Once the computation of posteriors for possibly repeating n-grams is finished, we use the algorithm described in (Allauzen et al., 2004) to construct a weighted factor automaton C mapping all n-grams observed in A to their expected counts, i.e. $\forall \pi$ in C, $i[\pi] = x \implies w[\pi] = c(x|A)$. We use P' and C to construct another weighted factor automaton P mapping all n-grams observed in A to their posterior probabilities, i.e. $\forall \pi$ in P, $i[\pi] = x \implies w[\pi] = p(x|A)$. First we remove the n-grams accepted by P' from C using the difference operation (Mohri, 2009),

$$C' = C - \mathrm{RmWeight}(P')$$

then take the union of the remaining automaton C' and P', and finally optimize the result by removing ε-transitions, determinizing and minimizing

$$P = \mathrm{Min}(\mathrm{Det}(\mathrm{RmEps}(C' \oplus P'))).$$

4 Experiments and Discussion

In this section we provide experiments comparing the performance of Algorithm 2 with Algorithm 1 as well as a baseline algorithm based on the approach of (Tromble et al., 2008). All algorithms were implemented in C++ using the OpenFst Library (Allauzen et al., 2007). Algorithm 1 implementation is a thin wrapper around the reference implementation. All experiments were conducted on the 88K ASR lattices (total size: #states + #arcs = 33M, disk size: 481MB) generated from the training subset of the IARPA Babel Turkish language pack, which includes 80 hours of conversational telephone speech. Lattices were generated with a speaker dependent DNN ASR system that was trained on the same data set using IBM's Attila toolkit (Soltau et al., 2010). All lattices were pruned to a logarithmic beam width of 5.

Figure 1 gives a scatter plot of the posterior probability computation time vs. the number of lattice n-grams (up to 5-grams) where each point

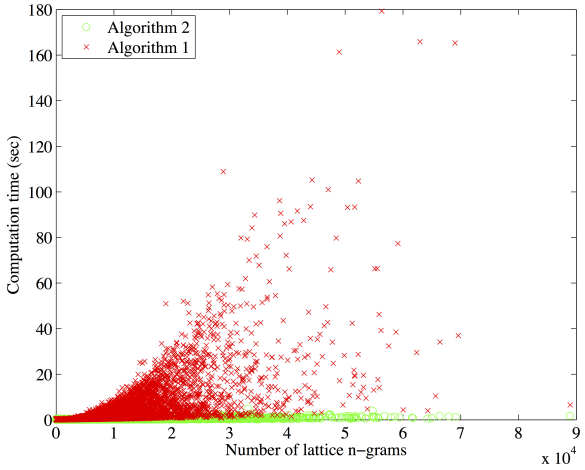

Figure 1: Runtime comparison

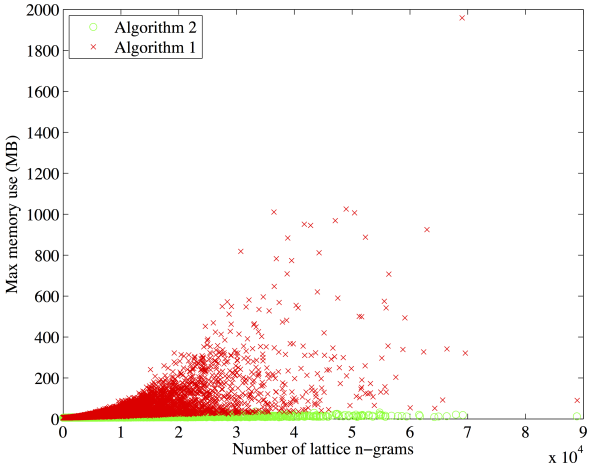

Figure 2: Memory use comparison

Table 2: Runtime Comparison

Max n-gram length	1	2	3	4	5	6	10	all
$\log_{10}(\#n\text{-grams})$	3.0	3.8	4.2	4.5	4.8	5.1	6.3	11.2
Baseline (sec)	5	15	32	69	147	311	5413	-
Algorithm 1 (sec)	0.5	0.6	0.9	1.6	3.9	16	997	-
Algorithm 2 (sec)	0.7	0.8	0.9	1.1	1.2	1.3	1.7	1.0
Expected Count (sec)	0.3	0.4	0.5	0.6	0.7	0.8	1.0	0.5

represents one of the 88K lattices in our data set. Similarly, Figure 2 gives a scatter plot of the maximum memory used by the program (maximum resident set size) during the computation of posteriors vs. the number of lattice n-grams (up to 5-grams). Algorithm 2 requires significantly less resources, particularly in the case of larger lattices with a large number of unique n-grams.

To better understand the runtime characteristics of Algorithms 1 and 2, we conducted a small experiment where we randomly selected 100 lattices (total size: #states + #arcs = 81K, disk size: 1.2MB) from our data set and analyzed the relation between the runtime and the maximum n-gram length N. Table 2 gives a runtime comparison between the baseline posterior computation algorithm described in (Tromble et al., 2008), Algorithm 1, Algorithm 2 and the expected count computation algorithm of (Allauzen et al., 2004). The baseline method computes posteriors separately for each n-gram by intersecting the lattice with an automaton accepting only the paths including that n-gram and computing the total weight of the resulting automaton in log semiring. Runtime complexities of the baseline method and Algorithm 1 are exponential in N due to the explicit enumeration of n-grams and we can clearly see this trend in the 3rd and 4th rows of Table 2. Algorithm 2 (5th row) takes advantage of the WFSA based expected count computation algorithm (6th row) to do most of the work for long n-grams, hence does not suffer from the same exponential growth. Notice the drops in the runtimes of Algorithm 2 and the WFSA based expected count computation algorithm when all n-grams are included into the computation regardless of their length. These drops are due to the expected count computation algorithm that processes all n-grams simultaneously using WFSA operations. Limiting the maximum n-gram length requires pruning long n-grams, which in general can increase the sizes of intermediate WFSAs used in computation and result in longer runtimes as well as larger outputs.

When there is no limit on the maximum n-gram length, the output of Algorithm 2 is a weighted factor automaton mapping each factor to its posterior. Table 3 compares the construction and storage requirements for posterior factor automata with similar factor automata structures. We use the approach described in (Allauzen et al., 2004) for constructing both the unweighted and the expected count factor automata. We construct the unweighted factor automata by first removing the weights on the input lattices and then applying the determinization operation on the tropical semiring so that path weights are not added together. The storage requirements of the posterior factor automata produced by Algorithm 2 is similar to those of the expected count factor automata. Unweighted factor automata, on the other hand, are significantly more compact than their weighted counterparts even though they accept the same set of strings. This difference in size is due to accommodating path weights which in general can significantly impact the effectiveness of automata determinization and minimization.

5 Related Work

Efficient computation of n-gram expected counts from weighted automata was first addressed in (Allauzen et al., 2003) in the context of estimating n-gram language model statistics from ASR lattices. Expected counts for all n-grams of interest observed in the input automaton are computed by composing the input with a simple counting transducer, projecting on the output side, and removing ε-transitions. The weight associated by the resulting WFSA to each n-gram it accepts is simply the expected count of that n-gram in the input automaton. Construction of such an automaton for all substrings (factors) of the input automaton was later explored in (Allauzen et al., 2004) in the con-

Table 3: Factor Automata Comparison

FA Type	Unweighted	Expected Count	Posterior
#states + #arcs (M)	16	20	21
On disk size (MB)	219	545	546
Runtime (min)	5.5	11	22

text of building an index for spoken utterance retrieval (SUR) (Saraclar and Sproat, 2004). This is the approach used for constructing the weighted factor automaton C in Algorithm 2. While expected count works well in practice for ranking spoken utterances containing a query term, posterior probability is in theory a better metric for this task. The weighted factor automaton P produced by Algorithm 2 can be used to construct an SUR index weighted with posterior probabilities.

The problem of computing n-gram posteriors from lattices was first addressed in (Tromble et al., 2008) in the context of lattice-based MBR for SMT. This is the baseline approach used in our experiments and it consists of building a separate FSA for each n-gram of interest and intersecting this automaton with the input lattice to discard those paths that do not include that n-gram and summing up the weights of remaining paths. The fundamental shortcoming of this approach is that it requires separate intersection and shortest distance computations for each n-gram. This shortcoming was first tackled in (Allauzen et al., 2010) by introducing a counting transducer for simultaneous computation of posteriors for all n-grams of order n in a lattice. This transducer works well for unigrams since there is a relatively small number of unique unigrams in a lattice. However, it is less efficient for n-grams of higher orders. This inefficiency was later addressed in (Blackwood et al., 2010) by employing n-gram mapping transducers to transduce the input lattices to n-gram lattices of order n and computing unigram posteriors on the higher order lattices. Algorithm 1 was described in (de Gispert et al., 2013) as a fast alternative to counting transducers. It is a lattice specialization of a more general algorithm for computing n-gram posteriors from a hypergraph in a single inside pass (DeNero et al., 2010). While this algorithm works really well for relatively short n-grams, its time and space requirements scale exponentially with the maximum n-gram length. Algorithm 2 builds upon this algorithm by exploiting the equivalence of expected counts and posteriors for non-repeating n-grams and eliminating the costly posterior computation operation for most n-grams in the input lattice.

6 Conclusion

We have described an efficient algorithm for computing n-gram posteriors from an input lattice and constructing an efficient and compact data structure for storing and retrieving them. The runtime and memory requirements of the proposed algorithm grow linearly with the length of the n-grams as opposed to the exponential growth observed with the original algorithm we are building upon. This is achieved by limiting the posterior computation to only those n-grams that may repeat on some path of the input lattice and using the relatively cheaper expected count computation algorithm for the rest. This filtering of n-grams introduces a slight bookkeeping overhead over the baseline algorithm but in return dramatically reduces the runtime and memory requirements for long n-grams.

Acknowledgments

The authors would like to thank Cyril Allauzen and Graeme W. Blackwood for helpful discussions. This work uses IARPA-babel105b-v0.4 Turkish full language pack from the IARPA Babel Program language collection and is supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD/ARL) contract number W911NF-12-C-0012. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 40–47, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cyril Allauzen, Mehryar Mohri, and Murat Saraclar. 2004. General indexation of weighted automata: Application to spoken utterance retrieval. In *HLT-NAACL Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval*, pages 33–40, Boston, MA, USA.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer. `http://www.openfst.org`.

Cyril Allauzen, Shankar Kumar, Wolfgang Macherey, Mehryar Mohri, and Michael Riley. 2010. Expected sequence similarity maximization. In *HLT-NAACL*, pages 957–965. The Association for Computational Linguistics.

Graeme Blackwood, Adrià de Gispert, and William Byrne. 2010. Efficient path counting transducers for minimum bayes-risk decoding of statistical machine translation lattices. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 27–32, Uppsala, Sweden, July. Association for Computational Linguistics.

Dogan Can and Shrikanth Narayanan. 2013. On the computation of document frequency statistics from spoken corpora using factor automata. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association*, pages 6–10, Lyon, France.

Ciprian Chelba, Timothy J. Hazen, and Murat Saraclar. 2008. Retrieval and browsing of spoken content. *Signal Processing Magazine, IEEE*, 25(3):39–49, May.

Adrià de Gispert, Graeme Blackwood, Gonzalo Iglesias, and William Byrne. 2013. N-gram posterior probability confidence measures for statistical machine translation: an empirical study. *Machine Translation*, 27(2):85–114.

John DeNero, Shankar Kumar, Ciprian Chelba, and Franz Och. 2010. Model combination for machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 975–983, Stroudsburg, PA, USA. Association for Computational Linguistics.

Vaibhava Goel and William J Byrne. 2000. Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135.

Damianos Karakos, Mark Dredze, Ken Ward Church, Aren Jansen, and Sanjeev Khudanpur. 2011. Estimating document frequencies in a speech corpus. In David Nahamoo and Michael Picheny, editors, *ASRU*, pages 407–412. IEEE.

Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. 2007. Factor automata of automata and applications. In *Implementation and Application of Automata*, pages 168–179. Springer.

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 213–254. Springer Berlin Heidelberg.

Murat Saraclar and Richard Sproat. 2004. Lattice-based search for spoken utterance retrieval. In *Proc. HLT-NAACL*, pages 129–136, Boston, MA, USA.

Hagen Soltau, George Saon, and Brian Kingsbury. 2010. The ibm attila speech recognition toolkit. In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 97–102, Dec.

Roy W Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. 2008. Lattice minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 620–629. Association for Computational Linguistics.