Generalized Algorithms for Constructing Statistical Language Models

Cyril Allauzen, Mehryar Mohri, Brian Roark
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932, USA
{allauzen,mohri,roark}@research.att.com

Abstract

Recent text and speech processing applications such as speech mining raise new and more general problems related to the construction of language models. We present and describe in detail several new and efficient algorithms to address these more general problems and report experimental results demonstrating their usefulness. We give an algorithm for computing efficiently the expected counts of any sequence in a word lattice output by a speech recognizer or any arbitrary weighted automaton; describe a new technique for creating exact representations of n-gram language models by weighted automata whose size is practical for offline use even for a vocabulary size of about 500,000 words and an n-gram order $n = 6$; and present a simple and more general technique for constructing class-based language models that allows each class to represent an arbitrary weighted automaton. An efficient implementation of our algorithms and techniques has been incorporated in a general software library for language modeling, the GRM Library, that includes many other text and grammar processing functionalities.

1 Motivation

Statistical language models are crucial components of many modern natural language processing systems such as speech recognition, information extraction, machine translation, or document classification. In all cases, a language model is used in combination with other information sources to rank alternative hypotheses by assigning them some probabilities. There are classical techniques for constructing language models such as n-gram models with various smoothing techniques (see Chen and Goodman (1998) and the references therein for a survey and comparison of these techniques).

In some recent text and speech processing applications, several new and more general problems arise that are related to the construction of language models. We present new and efficient algorithms to address these more general problems.

Counting. Classical language models are constructed by deriving statistics from large input texts. In speech mining applications or for adaptation purposes, one often needs to construct a language model based on the output of a speech recognition system. But, the output of a recognition system is not just text. Indeed, the word error rate of conversational speech recognition systems is still too high in many tasks to rely only on the one-best output of the recognizer. Thus, the word lattice output by speech recognition systems is used instead because it contains the correct transcription in most cases.

A word lattice is a weighted finite automaton (WFA) output by the recognizer for a particular utterance. It contains typically a very large set of alternative transcription sentences for that utterance with the corresponding weights or probabilities. A necessary step for constructing a language model based on a word lattice is to derive the statistics for any given sequence from the lattices or WFAs output by the recognizer. This cannot be done by simply enumerating each path of the lattice and counting the number of occurrences of the sequence considered in each path since the number of paths of even a small automaton may be more than four billion. We present a simple and efficient algorithm for computing the *expected count* of any given sequence in a WFA and report experimental results demonstrating its efficiency.

Representation of language models by WFAs. Classical n-gram language models admit a natural representation by WFAs in which each state encodes a left context of width less than n. However, the size of that representation makes it impractical for offline optimizations such as those used in large-vocabulary speech recognition or general information extraction systems. Most offline representations of these models are based instead on an approximation to limit their size. We describe a new technique for creating an *exact* representation of n-gram language models by WFAs whose size is practical for offline use even in tasks with a vocabulary size of about 500,000 words and for $n = 6$.

Class-based models. In many applications, it is natural and convenient to construct class-based language models, that is models based on classes of words (Brown et al., 1992). Such models are also often more robust since they may include words that belong to a class but that were not found in the corpus. Classical class-based models are based on simple classes such as a list of words. But new clustering algorithms allow one to create more general and more complex classes that may be regular languages. Very large and complex classes can also be defined using regular expressions. We present a simple and more general approach to class-based language models based on general weighted context-dependent rules (Kaplan and Kay, 1994; Mohri and Sproat, 1996). Our approach allows us to deal efficiently with more complex classes such as weighted regular languages.

We have fully implemented the algorithms just mentioned and incorporated them in a general software library for language modeling, the GRM Library, that includes many other text and grammar processing functionalities (Allauzen et al., 2003). In the following, we will present in detail these algorithms and briefly describe the corresponding GRM utilities.

2 Preliminaries

Definition 1 *A system* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is a* semiring *(Kuich and Salomaa, 1986) if:* $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid with identity element* $\overline{0}$*;* $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid with identity element* $\overline{1}$*;* $\otimes$ *distributes over* $\oplus$*; and* $\overline{0}$ *is an annihilator for* $\otimes$*: for all* $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$.

Thus, a semiring is a ring that may lack negation. Two semirings often used in speech processing are: the *log semiring* $\mathcal{L} = (\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$ (Mohri, 2002) which is isomorphic to the familiar real or probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$ via a log morphism with, for all $a, b \in \mathbb{R} \cup \{\infty\}$:

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

and the convention that: $\exp(-\infty) = 0$ and $-\log(0) = \infty$, and the *tropical semiring* $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ which can be derived from the log semiring using the Viterbi approximation.

Definition 2 *A* weighted finite-state transducer T *over a semiring* $\mathbb{K}$ *is an 8-tuple* $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ *where:* Σ *is the finite input alphabet of the transducer;* Δ *is the finite output alphabet;* Q *is a finite set of states;* $I \subseteq Q$ *the set of initial states;* $F \subseteq Q$ *the set of final states;* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$ *a finite set of transitions;* $\lambda : I \rightarrow \mathbb{K}$ *the initial weight function; and* $\rho : F \rightarrow \mathbb{K}$ *the final weight function mapping* F *to* $\mathbb{K}$.

A *Weighted automaton* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ is defined in a similar way by simply omitting the output labels. We denote by $L(A) \subseteq \Sigma^*$ the set of strings accepted by an automaton A and similarly by $L(X)$ the strings described by a regular expression X.

Given a transition $e \in E$, we denote by $i[e]$ its input label, $p[e]$ its origin or previous state and $n[e]$ its destination state or next state, $w[e]$ its weight, o[e] its output label (transducer case). Given a state $q \in Q$, we denote by $E[q]$ the set of transitions leaving q.

A *path* $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. We extend n and p to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. A cycle π is a path whose origin and destination states coincide: $n[\pi] = p[\pi]$. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, q')$ and $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and output label y (transducer case). These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. The labeling functions i (and similarly o) and the weight function w can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent transitions, and the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $i[\pi] = i[e_1] \cdots i[e_k]$, $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. We also extend w to any finite set of paths Π by setting: $w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi]$. The output weight associated by A to each input string $x \in \Sigma^*$ is:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

$[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$. Similarly, the output weight associated by a transducer T to a pair of input-output string (x, y) is:

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

$[\![T]\!](x, y) = \overline{0}$ when $P(I, x, y, F) = \emptyset$. A *successful path* in a weighted automaton or transducer M is a path from an initial state to a final state. M is *unambiguous* if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x. Thus, an unambiguous transducer defines a function.

For any transducer T, denote by $\Pi_2(T)$ the automaton obtained by projecting T on its output, that is by omitting its input labels.

Note that the second operation of the tropical semiring and the log semiring as well as their identity elements are identical. Thus the weight of a path in an automaton A over the tropical semiring does not change if A is viewed as a weighted automaton over the log semiring or vice-versa.

3 Counting

This section describes a *counting* algorithm based on general weighted automata algorithms. Let $A = (Q, I, F, \Sigma, \delta, \sigma, \lambda, \rho)$ be an arbitrary weighted automaton over the probability semiring and let X be a regular expression defined over the alphabet Σ. We are interested in *counting* the occurrences of the sequences $x \in L(X)$ in A while taking into account the weight of the paths where they appear.

3.1 Definition

When A is deterministic and *pushed*, or stochastic, it can be viewed as a probability distribution P over all strings Σ^*.[1] The weight $[\![A]\!](u)$ associated by A to a string u is

[1]There exist a general weighted determinization and weight pushing algorithms that can be used to create a deterministic and pushed automaton equivalent to an input word or phone lattice (Mohri, 1997).

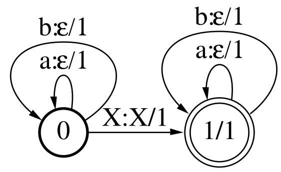

Figure 1: Counting weighted transducer T with $\Sigma = \{a, b\}$. The transition weights and the final weight at state 1 are all equal to 1.

then $P(u)$. Thus, we define the *count* of a sequence x in A, $c(x)$, as:

$$c(x) = \sum_{u \in \Sigma^*} |u|_x \, [\![A]\!](u)$$

where $|u|_x$ denotes the number of occurrences of x in the string u, i.e., the expected number of occurrences of x given A. More generally, we will define the count of x as above regardless of whether A is stochastic or not.

In most speech processing applications, A may be an acyclic automaton called a *phone* or a *word lattice* output by a speech recognition system. But our algorithm is general and does not assume A to be acyclic.

3.2 Algorithm

We describe our algorithm for computing the expected counts of the sequences $x \in L(X)$ and give the proof of its correctness.

Let S be the formal power series (Kuich and Salomaa, 1986) S over the probability semiring defined by $S = \Omega^* \times x \times \Omega^*$, where $x \in L(X)$.

Lemma 1 *For all* $\omega \in \Sigma^*$, $(S, \omega) = |\omega|_x$.

Proof. By definition of the multiplication of power series in the probability semiring:

$$\begin{aligned}(S, \omega) &= \sum_{u\,x\,v = \omega} (\Omega^*, u) \times (x, x) \times (\Omega^*, v) \\ &= 1_{\{(u,v):u\,x\,v=\omega\}} = |\omega|_x\end{aligned}$$

This proves the lemma. □

S is a rational power series as a product and closure of the polynomial power series Ω and x (Salomaa and Soittola, 1978; Berstel and Reutenauer, 1988). Similarly, since X is regular, the weighted transduction defined by $(\Sigma \times \{\epsilon\})^*(X \times X)(\Sigma \times \{\epsilon\})^*$ is rational. Thus, by the theorem of Schützenberger (Schützenberger, 1961), there exists a weighted transducer T defined over the alphabet Σ and the probability semiring realizing that transduction. Figure 1 shows the transducer T in the particular case of $\Sigma = \{a, b\}$.

Proposition 1 *Let A be a weighted automaton over the probability semiring, then:*

$$[\![\Pi_2(A \circ T)]\!](x) = c(x)$$

Proof. By definition of T, for any $\omega \in \Sigma^*$, $[\![T]\!](\omega, x) = (S, x)$, and by lemma 1, $[\![T]\!](\omega, x) = |\omega|_x$. Thus, by definition of composition:

$$\begin{aligned}[\![\Pi_2(A \circ T)]\!](x) &= \sum_{\pi \in P(I,F),\, \omega = i[\pi]} [\![A]\!](\omega) \times |\omega|_x \\ &= \sum_{\omega \in \Sigma^*} |\omega|_x \, [\![A]\!](\omega) = c(x)\end{aligned}$$

This ends the proof of the proposition. □

The proposition gives a simple algorithm for computing the expected counts of X in a weighted automaton A based on two general algorithms: composition (Mohri et al., 1996) and projection of weighted transducers. It is also based on the transducer T which is easy to construct. The size of T is in $O(|\Sigma| + |A_X|)$, where A_X is a finite automaton accepting X. With a lazy implementation of T, only one transition can be used instead of $|\Sigma|$, thereby reducing the size of the representation of T to $O(|A_X|)$.

The weighted automaton $B = \Pi_2(A \circ T)$ contains ϵ-transitions. A general ϵ-removal algorithm can be used to compute an equivalent weighted automaton with no ϵ-transition. The computation of $[\![B]\!](x)$ for a given x is done by composing B with an automaton representing x and by using a simple shortest-distance algorithm (Mohri, 2002) to compute the sum of the weights of all the paths of the result.

For numerical stability, implementations often replace probabilities with $-\log$ probabilities. The algorithm just described applies in a similar way by taking $-\log$ of the weights of T (thus all the weights of T will be zero in that case) and by using the log semiring version of composition and ϵ-removal.

3.3 GRM Utility and Experimental Results

An efficient implementation of the counting algorithm was incorporated in the GRM library (Allauzen et al., 2003). The GRM utility `grmcount` can be used in particular to generate a compact representation of the expected counts of the n-gram sequences appearing in a word lattice (of which a string encoded as an automaton is a special case), whose order is less or equal to a given integer. As an example, the following command line:

```
grmcount -n3 foo.fsm > count.fsm
```

creates an encoded representation `count.fsm` of the n-gram sequences, $n \leq 3$, which can be used to construct a trigram model. The encoded representation itself is also given as an automaton that we do not describe here.

The counting utility of the GRM library is used in a variety of language modeling and training adaptation tasks. Our experiments show that `grmcount` is quite efficient. We tested this utility with 41,000 weighted automata outputs of our speech recognition system for the same number of speech utterances. The total number of transitions of these automata was 18.8M. It took about 1h52m, including I/O, to compute the accumulated expected counts of all n-gram, $n \leq 3$, appearing in all these automata on a single processor of a 1GHz Intel Pentium processor Linux cluster with 2GB of memory and 256 KB cache. The time to compute these counts represents just $\frac{1}{50}$*th* of the total duration of the 41,000 speech utterances used in our experiment.

4 Representation of n-gram Language Models with WFAs

Standard smoothed n-gram models, including backoff (Katz, 1987) and interpolated (Jelinek and Mercer, 1980) models, admit a natural representation by WFAs in which each state encodes a conditioning history of length less than n. The size of that representation is often prohibitive. Indeed, the corresponding automaton may have $|\Sigma|^{n-1}$ states and $|\Sigma|^n$ transitions. Thus, even if the vocabulary size is just 1,000, the representation of a classical trigram model may require in the worst case up to one billion transitions. Clearly, this representation is even less adequate for realistic natural language processing applications where the vocabulary size is in the order of several hundred thousand words.

In the past, two methods have been used to deal with this problem. One consists of expanding that WFA on-demand. Thus, in some speech recognition systems, the states and transitions of the language model automaton are constructed as needed based on the particular input speech utterances. The disadvantage of that method is that it cannot benefit from offline optimization techniques that can substantially improve the efficiency of a recognizer (Mohri et al., 1998). A similar drawback affects other systems where several information sources are combined such as a complex information extraction system. An alternative method commonly used in many applications consists of constructing instead an approximation of that weighted automaton whose size is practical for offline optimizations. This method is used in many large-vocabulary speech recognition systems.

In this section, we present a new method for creating an *exact* representation of n-gram language models with WFAs whose size is practical even for very large-vocabulary tasks and for relatively high n-gram orders. Thus, our representation does not suffer from the disadvantages just pointed out for the two classical methods.

We first briefly present the classical definitions of n-gram language models and several smoothing techniques commonly used. We then describe a natural representation of n-gram language models using *failure transitions*. This is equivalent to the on-demand construction referred to above but it helps us introduce both the approximate solution commonly used and our solution for an exact offline representation.

4.1 Classical Definitions

In an n-gram model, the joint probability of a string $w_0 \dots w_k$ is given as the product of conditional probabilities:

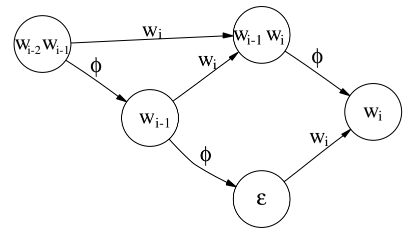

Figure 2: Representation of a trigram model with failure transitions.

$$\Pr(w_0 \dots w_k) = \prod_{i=0}^{k} \Pr(w_i|h_i) \quad (1)$$

where the conditioning history h_i consists of zero or more words immediately preceding w_i and is dictated by the order of the n-gram model.

Let $c(hw)$ denote the count of n-gram hw and let $\widehat{\Pr}(w|h)$ be the maximum likelihood probability of w given h, estimated from counts. $\widehat{\Pr}$ is often adjusted to reserve some probability mass for unseen n-gram sequences. Denote by $\widetilde{\Pr}(w|h)$ the adjusted conditional probability. Katz or absolute discounting both lead to an adjusted probability $\widetilde{\Pr}$.

For all n-grams $h = wh'$ where $h \in \Sigma^k$ for some $k \geq 1$, we refer to h' as the backoff n-gram of h. Conditional probabilities in a backoff model are of the form:

$$\Pr(w|h) = \begin{cases} \widetilde{\Pr}(w|h) & \text{if } c(hw) > 0 \\ \alpha_h \Pr(w|h') & \text{otherwise} \end{cases} \quad (2)$$

where α_h is a factor that ensures a normalized model. Conditional probabilities in a deleted interpolation model are of the form:

$$\Pr(w|h) = \begin{cases} (1-\alpha_h)\widehat{\Pr}(w|h) + \alpha_h \Pr(w|h') & \text{if } c(hw) > 0 \\ \alpha_h \Pr(w|h') & \text{otherwise} \end{cases} \quad (3)$$

where α_h is the mixing parameter between zero and one.

In practice, as mentioned before, for numerical stability, $-\log$ probabilities are used. Furthermore, due the Viterbi approximation used in most speech processing applications, the weight associated to a string x by a weighted automaton representing the model is the minimum weight of a path labeled with x. Thus, an n-gram language model is represented by a WFA over the tropical semiring.

4.2 Representation with Failure Transitions

Both backoff and interpolated models can be naturally represented using *default* or *failure transitions*. A failure transition is labeled with a distinct symbol ϕ. It is the default transition taken at state q when q does not admit an outgoing transition labeled with the word considered. Thus, failure transitions have the semantics of *otherwise*.

The set of states of the WFA representing a backoff or interpolated model is defined by associating a state q_h to each sequence of length less than n found in the corpus:

$$Q = \{q_h : |h| < n \text{ and } c(h) > 0\}$$

Its transition set E is defined as the union of the following set of failure transitions:

$$\{(q_{wh'}, \phi, -\log(\alpha_h), q_{h'}) : q_{wh'} \in Q\}$$

and the following set of regular transitions:

$$\{(q_h, w, -\log(\Pr(w|h)), n_{hw}) : q_{wh} \in Q\}$$

where n_{hw} is defined by:

$$n_{hw} = \begin{cases} q_{hw} & \text{if } 0 < |hw| < n \\ q_{h'w} & \text{if } |hw| = n \text{ where } h = w'h' \end{cases} \quad (4)$$

Figure 2 illustrates this construction for a trigram model. Treating ϵ-transitions as regular symbols, this is a deterministic automaton. Figure 3 shows a complete Katz backoff bigram model built from counts taken from the following toy corpus and using failure transitions:

⟨s⟩ b a a a a ⟨/s⟩
⟨s⟩ b a a a a ⟨/s⟩
⟨s⟩ a ⟨/s⟩

where ⟨s⟩ denotes the start symbol and ⟨/s⟩ the end symbol for each sentence. Note that the start symbol ⟨s⟩ does not label any transition, it encodes the history ⟨s⟩. All transitions labeled with the end symbol ⟨/s⟩ lead to the single final state of the automaton.

4.3 Approximate Offline Representation

The common method used for an offline representation of an n-gram language model can be easily derived from the representation using failure transitions by simply replacing each ϕ-transition by an ϵ-transition. Thus, a transition that could only be taken in the absence of any other alternative in the exact representation can now be taken regardless of whether there exists an alternative transition. Thus the approximate representation may contain paths whose weight does not correspond to the exact probability of the string labeling that path according to the model.

Consider for example the start state in figure 3, labeled with ⟨s⟩. In a failure transition model, there exists only one path from the start state to the state labeled a, with a cost of 1.108, since the ϕ transition cannot be traversed with an input of a. If the ϕ transition is replaced by an ϵ transition, there is a second path to the state labeled a – taking the ϵ-transition to the history-less state, then the a transition out of the history-less state. This path is not part of the probabilistic model – we shall refer to it as an *invalid path*. In this case, there is a problem, because the

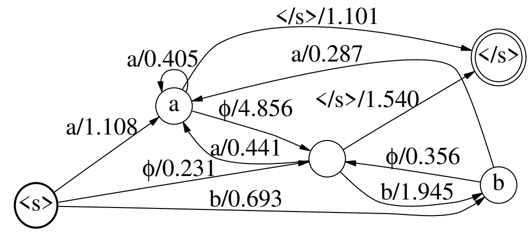

Figure 3: Example of representation of a bigram model with failure transitions.

cost of the invalid path to the state – the sum of the two transition costs (0.672) – is lower than the cost of the true path. Hence the WFA with ϵ-transitions gives a lower cost (higher probability) to all strings beginning with the symbol a. Note that the invalid path from the state labeled ⟨s⟩ to the state labeled b has a higher cost than the correct path, which is not a problem in the tropical semiring.

4.4 Exact Offline Representation

This section presents a method for constructing an exact offline representation of an n-gram language model whose size remains practical for large-vocabulary tasks.

The main idea behind our new construction is to modify the topology of the WFA to remove any path containing ϵ-transitions whose cost is lower than the correct cost associated by the model to the string labeling that path. Since, as a result, the low cost path for each string will have the correct cost, this will guarantee the correctness of the representation in the tropical semiring.

Our construction admits two parts: the detection of the invalid paths of the WFA, and the modification of the topology by splitting states to remove the invalid paths.

To detect invalid paths, we determine first their initial non-ϵ transitions. Let E_ϵ denote the set of ϵ-transitions of the original automaton. Let P_q be the set of all paths $\pi = e_1 \ldots e_k \in (E - E_\epsilon)^k$, $k > 0$, leading to state q such that for all i, $i = 1 \ldots k$, $p[e_i]$ is the destination state of some ϵ-transition.

Lemma 2 *For an n-gram language model, the number of paths in P_q is less than the n-gram order: $|P_q| < n$.*

Proof. For all $\pi_i \in P_q$, let $\pi_i = \pi'_i e_i$. By definition, there is some $e'_i \in E_\epsilon$ such that $n[e'_i] = p[e_i] = q_{h_i}$. By definition of ϵ-transitions in the model, $|h_i| < n - 1$ for all i. It follows from the definition of regular transitions that $n[e_i] = q_{h_i w} = q$. Hence, $h_i = h_j = h$, i.e. $e_i = e_j = e$, for all $\pi_i, \pi_j \in P_q$. Then, $P_q = \{\pi e : \pi \in P_{q_h}\} \cup \{e\}$. The history-less state has no incoming non-ϵ paths, therefore, by recursion, $|P_q| = |P_{q_h}| + 1 = |hw| < n$. □

We now define transition sets $D_{qq'}$ (originally empty) following this procedure: for all states $r \in Q$ and all $\pi = e_1 \ldots e_k \in P_r$, if there exists another path $\pi' = e\pi$ such that $e \in E_\epsilon$ and $i[\pi'] = i[\pi]$, and either (i) $n[\pi'] = n[\pi]$ and $w[e\pi] < w[\pi']$ or (ii) there exists $e' \in E_\epsilon$ such that $p[e'] = n[\pi']$ and $n[e'] = n[\pi]$ and $w[e\pi] < w[\pi' e']$, then we add e_1 to the set: $D_{p[\pi]p[\pi']} \leftarrow D_{p[\pi]p[\pi']} \cup \{e_1\}$. See

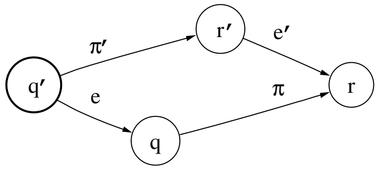

Figure 4: The path $e\pi$ is invalid if $i[e] = \epsilon$, $i[\pi] = i[\pi']$, $\pi \in P_r$, and either (i) $r' = r$ and $w[e\pi] < w[\pi']$ or (ii) $i[e'] = \epsilon$ and $w[e\pi] < w[\pi' e']$.

figure 4 for an illustration of this condition. Using this procedure, we can determine the set:

$$\tilde{E}[q] = \{e \in E[q] : \exists q', e \in D_{qq'}\}.$$

This set provides the first non-ϵ transition of each invalid path. Thus, we can use these transitions to eliminate invalid paths.

Proposition 2 *The cost of the construction of $\tilde{E}[q]$ for all $q \in Q$ is $n^2|\Sigma||Q|$, where n is the n-gram order.*

Proof. For each $q \in Q$ and each $\pi \in P_q$, there are at most $|\Sigma|$ possible states q' such that for some $e \in E_\epsilon$, $p[e] = q'$ and $n[e] = q$. It is trivial to see from the proof of lemma 2 that the maximum length of π is n. Hence, the cost of finding all π' for a given π is $n|\Sigma|$. Therefore, the total cost is $n^2|\Sigma||Q|$. □

For all non-empty $\tilde{E}[q]$, we create a new state $\tilde{q}$ and for all $e \in \tilde{E}[q]$ we set $p[e] = \tilde{q}$. We create a transition $(\tilde{q}, \epsilon, 0, q)$, and for all $e \in E - E_\epsilon$ such that $n[e] = q$, we set $n[e] = \tilde{q}$. For all $e \in E_\epsilon$ such that $n[e] = q$ and $|D_{qp[e]}| = 0$, we set $n[e] = \tilde{q}$. For all $e \in E_\epsilon$ such that $n[e] = q$ and $|D_{qp[e]}| > 0$, we create a new intermediate backoff state $\bar{q}$ and set $n[e] = \bar{q}$; then for all $e' \in E[\tilde{q}]$, if $e' \notin D_{qp[e]}$, we add a transition $\hat{e} = (\bar{q}, i[e'], w[e'], n[e'])$ to E.

Proposition 3 *The WFA over the tropical semiring modified following the procedure just outlined is equivalent to the exact online representation with failure transitions.*

Proof. Assume that there exists a string s for which the WFA returns a weight $\tilde{w}(s)$ less than the correct weight $w(s)$ that would have been assigned to s by the exact online representation with failure transitions. We will call an ϵ-transition e_i within a path $\pi = e_1 \dots e_k$ *invalid* if the next non-ϵ transition e_j, $j > i$ has the label w, and there is a transition e with $p[e] = p[e_i]$ and $i[e] = w$. Let π be a path through the WFA such that $i[\pi] = s$ and $w[\pi] = \tilde{w}(s)$, and π has the least number of invalid ϵ-transitions of all paths labeled with s with weight $\tilde{w}(s)$. Let e_i be the last invalid ϵ-transition taken in path π. Let π' be the valid path leaving $p[e_i]$ such that $i[\pi'] = i[e_{i+1} \dots e_k]$. $w[\pi'] > w[e_i \dots e_k]$, otherwise there would be a path with fewer invalid ϵ-transitions with weight $\tilde{w}(s)$. Let r be the first state where paths π' and $e_{i+1} \dots e_k$ intersect. Then $r = n[e_j]$ for some $j > i$. By definition, $e_{i+1} \dots e_j \in P_r$, since intersection will occur before any ϵ transitions are traversed in π. Then it must be the case that $e_{i+1} \in D_{n[e_i]p[e_i]}$, requiring the path to be removed from the WFA. This is a contradiction. □

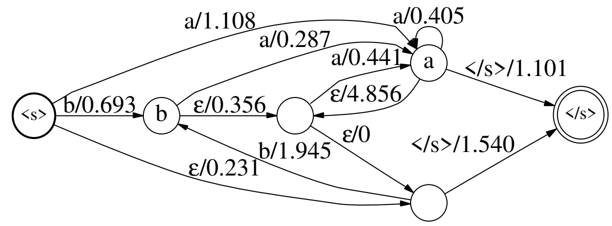

Figure 5: Bigram model encoded exactly with ϵ-transitions.

4.5 GRM Utility and Experimental Results

Note that some of the new intermediate backoff states ($\bar{q}$) can be fully or partially merged, to reduce the space requirements of the model. Finding the optimal configuration of these states, however, is an NP-hard problem. For our experiments, we used a simple greedy approach to sharing structure, which helped reduce space dramatically.

Figure 5 shows our example bigram model, after application of the algorithm. Notice that there are now two history-less states, which correspond to q and $\tilde{q}$ in the algorithm (no $\bar{q}$ was required). The start state backs off to q, which does not include a transition to the state labeled a, thus eliminating the invalid path.

Table 1 gives the sizes of three models in terms of transitions and states, for both the failure transition and ϵ-transition encoding of the model. The DARPA North American Business News (NAB) corpus contains 250 million words, with a vocabulary of 463,331 words. The Switchboard training corpus has 3.1 million words, and a vocabulary of 45,643. The number of transitions needed for the exact offline representation in each case was between 2 and 3 times the number of transitions used in the representation with failure transitions, and the number of states was less than twice the original number of states. This shows that our technique is practical even for very large tasks.

Efficient implementations of model building algorithms have been incorporated into the GRM library. The GRM utility `grmmake` produces basic backoff models, using Katz or Absolute discounting (Ney et al., 1994) methods, in the topology shown in figure 3, with ϵ-transitions in the place of failure transitions. The utility `grmshrink` removes transitions from the model according to the shrinking methods of Seymore and Rosenfeld (1996) or Stolcke (1998). The utility `grmconvert` takes a backoff model produced by `grmmake` or `grmshrink` and converts it into an exact model using either failure transitions or the algorithm just described. It also converts the model to an interpolated model for use in the tropical semiring. As an example, the following command line:

```
grmmake -n3 counts.fsm > model.fsm
```

creates a basic Katz backoff trigram model from the

Corpus	Model order	ϕ-representation arcs	ϕ-representation states	exact offline arcs	exact offline states
NAB	3-gram	102752	16838	303686	19033
SWBD	3-gram	2416	475	5499	573
SWBD	6-gram	15430	6295	54002	12374

Table 1: Size of models (in thousands) built from the NAB and Switchboard corpora, with failure transitions ϕ versus the exact offline representation.

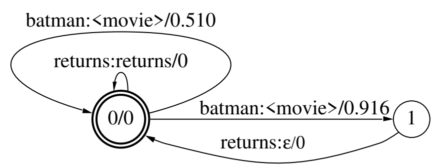

Figure 6: Weighted transducer T obtained from the compilation of context-dependent rewrite rules.

counts produced by the command line example in the earlier section. The command:

```
grmshrink -c1 model.fsm > m.s1.fsm
```

shrinks the trigram model using the weighted difference method (Seymore and Rosenfeld, 1996) with a threshold of 1. Finally, the command:

```
grmconvert -tfail m.s1.fsm > f.s1.fsm
```

outputs the model represented with failure transitions.

5 General class-based language modeling

Standard class-based or phrase-based language models are based on simple classes often reduced to a short list of words or expressions. New spoken-dialog applications require the use of more sophisticated classes either derived from a series of regular expressions or using general clustering algorithms. Regular expressions can be used to define classes with an infinite number of elements. Such classes can naturally arise, e.g., dates form an infinite set since the year field is unbounded, but they can be easily represented or approximated by a regular expression. Also, representing a class by an automaton can be much more compact than specifying them as a list, especially when dealing with classes representing phone numbers or a list of names or addresses.

This section describes a simple and efficient method for constructing class-based language models where each class may represent an arbitrary (weighted) regular language.

Let $c_1, c_2, \ldots, c_n$ be a set of n classes and assume that each class c_i corresponds to a stochastic weighted automaton A_i defined over the log semiring. Thus, the weight $[\![A_i]\!](w)$ associated by A_i to a string w can be interpreted as $-\log$ of the conditional probability $P(w|c_i)$. Each class c_i defines a weighted transduction:

$$A_i \longrightarrow c_i$$

This can be viewed as a specific obligatory weighted context-dependent rewrite rule where the left and right

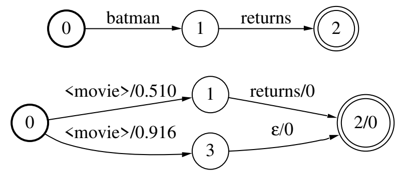

Figure 7: Corpora X and $\hat{X}$.

contexts are not restricted (Kaplan and Kay, 1994; Mohri and Sproat, 1996). Thus, the transduction corresponding to the class c_i can be viewed as the application of the following obligatory weighted rewrite rule:

$$A_i \to c_i/\epsilon \_\_ \epsilon$$

The direction of application of the rule, left-to-right or right-to-left, can be chosen depending on the task[2]. Thus, these n classes can be viewed as a set of batch rewrite rules (Kaplan and Kay, 1994) which can be compiled into weighted transducers. The utilities of the GRM Library can be used to compile such a batch set of rewrite rules efficiently (Mohri and Sproat, 1996).

Let T be the weighted transducer obtained by compiling the rules corresponding to the classes. The corpus can be represented as a finite automaton X. To apply the rules defining the classes to the input corpus, we just need to compose the automaton X with T and project the result on the output:

$$\hat{X} = \Pi_2(X \circ T)$$

$\hat{X}$ can be made stochastic using a pushing algorithm (Mohri, 1997). In general, the transducer T may not be unambiguous. Thus, the result of the application of the class rules to the corpus may not be a single text but an automaton representing a set of alternative sequences. However, this is not an issue since we can use the general counting algorithm previously described to construct a language model based on a weighted automaton. When $L = \cup_{i=1}^{n} L(A_i)$, the language defined by the classes, is a *code*, the transducer T is unambiguous.

Denote now by $\hat{G}$ the language model constructed from the new corpus $\hat{X}$. To construct our final class-based language model G, we simply have to compose $\hat{G}$ with T^{-1} and project the result on the output side:

$$G = \Pi_2(\hat{G} \circ T^{-1})$$

A more general approach would be to have two transducers T_1 and T_2, the first one to be applied to the corpus and the second one to the language model. In a probabilistic interpretation, T_1 should represent the probability distribution $P(c_i|w)$ and T_2 the probability distribution $P(w|c_i)$. By using $T_1 = T$ and $T_2 = T^{-1}$, we are in fact making the assumptions that the classes are equally probable and thus that $P(c_i|w) = P(w|c_i)/\Sigma_{j=1}^{n} P(w|c_j)$.

[2] The simultaneous case is equivalent to the left-to-right one here.

More generally, the weights of T_1 and T_2 could be the results of an iterative learning process. Note however that we are not limited to this probabilistic interpretation and that our approach can still be used if T_1 and T_2 do not represent probability distributions, since we can always push $\hat{X}$ and normalize G.

Example. We illustrate this construction in the simple case of the following class containing movie titles:

<movie>$= \{(\text{batman}, 0.6), (\text{batman returns}, 0.4)\}$

The compilation of the rewrite rule defined by this class and applied left to right leads to the weighted transducer T given by figure 6. Our corpus simply consists of the sentence “batman returns” and is represented by the automaton X given by figure 7. The corpus $\hat{X}$ obtained by composing X with T is given by figure 7.

6 Conclusion

We presented several new and efficient algorithms to deal with more general problems related to the construction of language models found in new language processing applications and reported experimental results showing their practicality for constructing very large models. These algorithms and many others related to the construction of weighted grammars have been fully implemented and incorporated in a general grammar software library, the GRM Library (Allauzen et al., 2003).

Acknowledgments

We thank Michael Riley for discussions and for having implemented an earlier version of the counting utility.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. GRM Library-Grammar Library. *http://www.research.att.com/sw/tools/grm*, AT&T Labs - Research.

Jean Berstel and Christophe Reutenauer. 1988. *Rational Series and Their Languages*. Springer-Verlag: Berlin-New York.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Stanley Chen and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report, TR-10-98, Harvard University.

Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3).

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 35(3):400–401.

Werner Kuich and Arto Salomaa. 1986. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany.

Mehryar Mohri and Richard Sproat. 1996. An Efficient Compiler for Weighted Rewrite Rules. In 34*th Meeting of the Association for Computational Linguistics (ACL ’96), Proceedings of the Conference, Santa Cruz, California*. ACL.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 1996. Weighted Automata in Text and Speech Processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language, Budapest, Hungary*. ECAI.

Mehryar Mohri, Michael Riley, Don Hindle, Andrej Ljolje, and Fernando C. N. Pereira. 1998. Full expansion of context-dependent networks in large vocabulary speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Mehryar Mohri. 1997. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2.

Mehryar Mohri. 2002. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modeling. *Computer Speech and Language*, 8:1–38.

Arto Salomaa and Matti Soittola. 1978. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag: New York.

Marcel Paul Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4.

Kristie Seymore and Ronald Rosenfeld. 1996. Scalable backoff language models. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274.