A Generalization and Improvement to PPM's "Blending"[1]

Suzanne Bunton

Technical Report UW-CSE-97-01-10

Department of Computer Science and Engineering
University of Washington

Abstract

The best-performing method in the data compression literature for computing probability estimates of sequences on-line using a suffix-tree model is the *blending* technique used by PPM. Blending can be viewed as a bottom-up recursive procedure for computing a *mixture*, barring one missing term for each level of the recursion, where a mixture is basically a weighted average of several probability estimates. We show by decomposition into an *inheritance evaluation time* and a *mixture weighting function* that mixtures generalize the techniques used in PPM variants. Doubly controlled experiments with our executable taxonomy of on-line sequence modeling algorithms and the Calgary Corpus demonstrate the impact of varying inheritance evaluation time, mixture weighting function, and including update exclusion.

Keywords: data compression, universal coding, on-line stochastic modeling, statistical inference, finite-state automata

[1]Portions of this paper also appear in Proceedings of the DCC, March 1997

Generalization and Improvement to PPM's "Blending"

Suzanne Bunton

The University of Washington

The best-performing method in the data compression literature for computing probability estimates of sequences on-line using a suffix-tree model is the *blending* technique used by PPM [CW84, Mof90]. Blending can be viewed as a bottom-up recursive procedure for computing a *mixture*, barring one missing term for each level of the recursion, where a mixture is basically a weighted average of several probability estimates. We shall show by decomposition that mixtures generalize the techniques used in DMC variants [CH87, TR93], as well as PPM variants, and thus these techniques, along with other variants of mixtures, are interchangeable.

1 Recursive Mixtures

We are concerned with estimating a probability $P_e(a_i|a_1a_2\cdots a_{i-1})$ using the frequencies stored at the *excited* states of a suffix-tree FSM (see Chapter 2 of [Bun96]), where the excited states are those states of the FSM whose associated conditioning context partitions contain the sequence $a_1a_2\cdots a_{i-1} \in A^*$. At any time, the excited states of a suffix-tree FSM are linked, at least abstractly, by an unbroken chain of suffix pointers, which, for a given state s, point to the state with the smallest conditioning context that properly contains the conditioning context of s.

Let s_0 and s_{-1} be the order 0 and order $\Leftrightarrow$1 states of a suffix tree, respectively, and define $\alpha(s)$ to be the number of times a novel event has occurred at a given state s. That is,

$$\alpha(s) = |\{a : \mathbf{count}[a, s, u(s)] > 0\}|,$$

where $a \in A$, the finite input alphabet, and $u : S \rightarrow \{0, 1\}$ selects between full-update frequencies given by **count**[-,s,0] and update-excluded[2] frequencies given by **count**[-,s,1] at state s. Thus,

$$u(s) = \begin{cases} 0 & \text{if } s = s', \text{ the selected state;} \\ X & \text{otherwise,} \end{cases}$$

where X is a global binary variable denoting whether *update exclusions* are enabled for the model. Note that $\mathbf{count}[a, s, 1] = 0 \Leftrightarrow \mathbf{count}[a, s, 0] = 0$, so the value of $u(s)$ does not affect the value of $\alpha(s)$. Let $\mathbf{count}(a, s) = \mathbf{count}[a, s, u(s)] + k$, where k is the initial frequency value (ideally, $k = 0$), and k is a global constant that remains fixed for the lifetime of the model. Lastly, let the node-count function $\mathbf{count} : S \rightarrow R$ be defined as follows:

$$\mathbf{count}(s) = \sum_{a:\mathbf{count}[a,s,u(s)]>0} \mathbf{count}(a, s).$$

[2] Chapter 5 of [Bun96] explains that the ability to dynamically select update-excluded frequencies or full-update frequencies on a per-state basis is required for correctly combining mixtures, update exclusion (introduced in [Mof90]), and state selection.

Given the above definitions, a simple bottom-up procedure for recursively computing a mixture that estimates the probability of a given event, $a_i = a$, starting from an excited state s, is

$$P_e(a|s,i) = \begin{cases} W(s) \cdot \frac{\mathbf{count}(a,s)}{\mathbf{count}(s)} + (1 \Leftrightarrow W(s)) \cdot P_e(a|\mathbf{suffix}(s), h(s,a,i)) & \text{if } s \neq s_{-1} \\ \frac{1}{|A|+1-\alpha(s_0)} & \text{otherwise} \end{cases}$$

where $0 \leq W(s) < 1$, and $h(s,a,i) \leq i$.

Assuming the mixture computation is initiated at the maximum-order excited state, this procedure computes a mixture of maximum-likelihood probability estimators for all excited states, except for the order $\Leftrightarrow$1 state, which contributes an explicitly assumed initial distribution that must be non-zero for all possible symbols in A.

The *inheritance evaluation time* $h(s,a,i)$ defines when, relative to the input sequence, the ancestor's contribution to the mixture, $P_e(a|\mathbf{suffix}(s), h(s,a,i))$, is computed. The recursive *mixture weighting function* $W()$ determines the degree of influence the ancestor's contribution will have relative to the contribution of the frequencies local to state s.

There are two essential questions that must be addressed when defining an effective mixture:

- How do we define the *mixture weighting function* $W()$?
- How do we define the *inheritance evaluation time* $h(s,a,i)$?

2 Mixture Weights

Our goal is to define an easily computable weighting function $W : S \rightarrow [0..1)$ that will cause $P_e(a|s)$ to assign the greatest likelihood to the currently scanned symbol, on average. This implies, for one thing, that our weighting function should assign a low value to $W(s)$ whenever it is likely that the currently scanned symbol corresponds to an event that has never occurred when s was excited, so that the weight $1 \Leftrightarrow W(s)$ of the ancestors' contribution to the mixture is relatively high. Thus the choice of weighting function reduces to a solution to an ancient problem—the "zero-frequency problem," or how to assign a likelihood to an event that has never occurred before—for which it is widely agreed that no principled solution exists, in the absence of *a priori* knowledge [WB91]. Therefore, the merit of any weighting function for a universal model is determined analytically by how the assumptions it imposes interact with other assumptions made in the model, and empirically by its performance on actual data.

Several approaches to solving the zero frequency problem, known as "escape" mechanisms, have been used successfully with PPM implementations. Four of the simplest and best-performing escape mechanisms are known in the literature as 'A,' 'B,' 'C,' and 'D' [WB91]. In this section, we shall show how these simple escape mechanisms correspond to different weighting functions $W(s)$. We introduce a general formula for $W(s)$ that relies upon global changes to the initial values of event frequencies to express each of these escape mechanisms exactly, and which allows efficient implementation of the mixture computation.

2.1 Mixture Weights with Variable Initial Frequencies

Each of the escape mechanisms 'A'–'D' can be described exactly as a general weighting function $W(s)$, where

$$W(s) = \frac{\mathbf{count}(s)}{\mathbf{count}(s) + \frac{\alpha(s)}{d(s)}},$$

if we let the escape mechanism determine the global constant k (which is ideally zero) such that $\mathbf{count}(a, s) = \mathbf{count}[a, s, u(s)] + k$ if $\mathbf{count}[a, s, u(s)] > 0$, and $\mathbf{count}(a, s) = 0$ if $\mathbf{count}[a, s, u(s)] = 0$. This way, the four escape mechanisms are given by the following assignments to $d(s)$ and initial frequency value k:

A:	$d(s) = \alpha(s)$	$k = 0$
B:	$d(s) = 1$,	$k = \Leftrightarrow 1$
C:	$d(s) = 1$,	$k = 0$
D:	$d(s) = 2$,	$k = \Leftrightarrow \frac{1}{2}$.

What assumptions about the input data do these choices of weighting formulas impose? Each of these weighting functions base the weights on the number of times in the past that a node s has "missed," that is, failed to assign a non-zero likelihood to the scanned symbol when excited. The key difference among the escape mechanisms is how much emphasis is placed on the predictions conditioned by excited low order states relative to the predictions conditioned by excited high order states.

In the general weighting formula for $W(s)$ above, more emphasis is placed on higher order states as $d(s)$ increases in numerical value. Thus, if we sort the escape mechanisms by increasing values of $d(s)$, we get 'B' $\preceq$ 'C' $\preceq$ 'A' $\prec$ 'D', when $\alpha(s) < 2$ and 'B' $\preceq$ 'C' $\prec$ 'D' $\preceq$ 'A', otherwise. With larger values of k, more emphasis is placed upon higher order states. Thus if we sort the escape mechanisms by increasing values of k, we get 'B' $\prec$ 'D' $\prec$ 'C' $\preceq$ 'A'. Clearly 'B' places the lowest relative emphasis on high order states, while 'A' tends to place the greatest emphasis on high order states. Mechanisms 'D' and 'C', which consistently and significantly outperform 'A' and 'B' in practice, are somewhere in the middle. Method 'D' systematically favors deterministic states (i.e., states that recognize only one input symbol—they are always among the highest order excited states), and tends to slightly outperform 'C'.

One clear benefit of these particular escape mechanisms is that they simplify the algebra required to compute the mixture. With escape formulas 'A'–'D', the general mixture formula becomes:

$$P_e(a|s,i) = \begin{cases} \frac{\mathbf{count}(a,s) + \left(\frac{\alpha(s)}{d(s)} \cdot P_e(a|\mathbf{suffix}(s), h(s,a,i))\right)}{\mathbf{count}(s) + \frac{\alpha(s)}{d(s)}} & \text{if } s \neq s_{-1} \\ \frac{1}{|A| + 1 - \alpha(s_0)} & \text{otherwise} \end{cases}$$

where $0 \leq W(s) < 1$.

2.2 Inherited Frequencies

In general, given any of the weighting formulae above, we can express the mixture as

$$P_e(a|s,i) = \begin{cases} \frac{\frac{\text{numerator}(W(s))}{\mathbf{count}(s)} \cdot \mathbf{count}(a,s) + I(a,s,i)}{\text{denominator}(W(s))} & \text{if } s \neq s_{-1} \\ \frac{1}{|A| + 1 - \alpha(s_0)} & \text{otherwise,} \end{cases}$$

where $I(a,s,i) = (\alpha(s)/d(s)) \cdot P_e(a|\textbf{suffix}(s), h(s,a,i))$. $I(a,s,i)$ is the *inherited frequency* for event $a|s$ at time i. The next section covers the computation of $I(a,s,i)$ and how different computation times affect the model.

3 Inheritance Evaluation Times

There is a spectrum of evaluation times for inherited frequencies, which denote when $\alpha(s)/d(s) \cdot P_e(a|\textbf{suffix}(s), h(s,a,i))$ is computed, relative to the lifetimes of state s and event $a|s$. We use *inheritance evaluation time* $h(s,a,i)$ to explicitly specify $P_e(a|\textbf{suffix}(s))$ as a function of the frequency data that are available at one of the following times:

- *inherit at model creation:* $P_e(a|\textbf{suffix}(s), h(s,a,i))$ is computed when the initial model consisting of state s_{-1} is created; $h(s,a,i) = 0$.
- *inherit at state creation:* $P_e(a|\textbf{suffix}(s), h(s,a,i))$ is computed when state s is added to the model; $h(s,a,i)$ equals the length of the input sequence that had been processed so far when s was added to the model.
- *inherit before novel event update:* $P_e(a|\textbf{suffix}(s), h(s,a,i))$ is computed when novel event $a|s$ first occurs, before its frequency is incremented; $h(s,a,i)$ equals the length of the input sequence that had been processed so far when a occurred for the first time when s was excited.
- *inherit at every event visit:* $P_e(a|\textbf{suffix}(s))$ is (re)computed each time event $a|s$ occurs; $h(s,a,i) = i$.

Inheritance evaluation time is a global option that remains fixed for the lifetime of the model. In the remainder of this work we will usually simplify the notation by making evaluation times i and $h(s,a,i)$ implicit, thus replacing $P_e(a|s,i)$, $I(a,s,i)$, and $P_e(a|\textbf{suffix}(s), h(s,a,i))$ with $P_e(a|s)$, $I(a,s)$, and $P_e(a|\textbf{suffix}(s))$ respectively.

There are two important features that must be considered in the choice of an inheritance evaluation time. The computational and memory cost of its implementation, and what the inheritance evaluation time assumes about the data, relative to the other times. Once the relative differences in inheritance evaluation times are understood, the model designer can intelligently trade off the appropriateness of the assumption about the target data versus the space and time requirements of an implementation.

3.1 Inheritance Evaluation Times in Practice

Probably the most natural time for computing inherited frequencies $I(a,s)$ is every single time the state s becomes excited: in this case, the mixture is simply a weighted average of the probabilities estimated from the current frequency data at each excited state. In [BCW90] this approach is called "full blending." However, computing such weighted averages is expensive, and computations cannot be reused between visits to a given set of excited states. Furthermore, there was no published evidence prior to this work that it produces better probability estimates—no published on-line algorithms use it. The remaining alternatives generally allow faster implementations.

At the other extreme, *inherit at model creation* corresponds to adding a constant assumed initial frequency distribution to the observed frequencies at any given node. This approach leads to simple analyses and does not affect asymptotic convergence.

Thus it is employed by most theoretical constructions that use information-theoretic state-selection. However, our experiments show that the probability estimates produced by using this inheritance evaluation time are not competitive with the other inheritance times considered here, even when they are combined with state selection.

The DMC algorithm, which originally used a binary alphabet, adds each new state to its model by "cloning" an eligible parent state. Each clone receives a scaled copy of the parent state's frequency distribution the moment it is added. Since, as we proved in Chapter 4 of [Bun96], the conditioning context relationship among clones and parent states is equivalent to that among suffixes in other suffix-tree models, *inherit at state creation* corresponds to the numerical aspects of cloning. For non-binary input alphabets and aggressive model growth heuristics, evaluating every symbol's inherited frequency at every new state is prohibitive, in terms of memory and computation costs, as was historically demonstrated with larger-alphabet parameterizations of the DMC algorithm. We include this inheritance time only for completeness, and do not evaluate its performance.

Overall, the best approach practically, and performance-wise, is the evaluation of inherited frequencies whenever a novel event a is seen at state s. This is similar to blending in PPM variants, and can also be used in lazy implementations of large-alphabet DMC variants.

3.2 The Significance of Inheritance Evaluation Time

Basically, inheritance evaluation times select the degree to which *recent vs. relatively historical* event frequencies that are conditioned by a given set of contexts predict the behavior of the events conditioned by proper subclasses of those contexts. In contrast, mixture weighting functions determine the degree to which *inherited vs. observed* event frequencies predict the behavior of events conditioned by a given set of contexts.

There are more direct and quantifiable ways of establishing how much more recent events should matter than events that happened long ago. For example, a sliding window of input history can be kept, and as sequence symbols that have passed through the buffer pass out of the buffer, the event frequencies originally incremented by these symbols can be decremented [Wil91]. Alternatively, at regular intervals, all the frequencies in the model can be scaled by a small constant, which would implement an exponential decay function [How93]. Or, the same process could be carried on locally, on a per-state basis, when the state's total frequency exceeded a threshold [Mof90].

However, regardless of whatever merit direct techniques for recency-weighting stored frequencies may have (none has been shown to consistently improve predictions of blended techniques), these approaches each add an additional feature to the model. Even without such an added feature, every on-line model must by default implement an inheritance evaluation time; and, the selection of an inheritance evaluation time should be made with some consideration of its appropriateness for the input data. In fact, we can compare the relative effect that different inheritance evaluation times have on model inferences by means of an analogy to family traditions for passing parental knowledge and experience down to children.

3.2.1 Some Intuition

Suppose instead of stochastic models, our suffix-trees represent family trees, where the nodes correspond to people (for simplicity, consider family members of only one sex), the events correspond to situations that the people may find themselves in (such as handling a bully, training a puppy, initiating courtship, buying a used car), and the suffix relationship corresponds to the parent relationship. Here, the "inheritance" received by children is parental knowledge, which is based upon the parent's past experiences plus the parent's own inheritance from the grandparent. Each inheritance evaluation time corresponds to a family tradition for passing knowledge down to children that is strictly maintained by the descendants in each family tree. Note that in this analogy, as in suffix-tree stochastic models, the *weight* that a child assigns to the advice received (inherited) from his parent, relative to his own experience, is a matter completely independent from *when* he receives advice.

The most conservative family tradition allows each parent to pass down only what was passed to him from his parent. This results in each child receiving only the ancient laws that trace back to the family's progenitor. In this tradition, children cannot benefit from their parent's (or grandparent's, ...) experience, and therefore must learn mostly from their own experience. Clearly younger children born to such traditions would have trouble competing with same-age children from more communicative families, although this competitive difference diminishes among older, more experienced children. This corresponds to *inherit at model creation*.

In contrast, the most liberal family tradition requires that every child, before handling any event in his lifetime, regardless of his own experience, listen to advice from his parent, based upon the parent's and other ancestor's current knowledge. The drawback of this approach is frustratingly high communication overhead. However, the benefit is that a parent is able to completely revise the poor advice on a given subject he may have given when he himself was relatively inexperienced. This corresponds to *inherit at every event visit*.

In the middle lies the approach that is taken naturally by most actual families: children consult parents when they face a completely novel situation, and rely increasingly on their own experience with each recurrence of the situation. This corresponds to *inherit at novel event occurrence*. Blending is a simplification of this approach: children consult parents for novel events but rely thereafter upon their own experience with each recurrence.

Last is the rather anxious approach in which the parent coaches each child on how to handle every imaginable situation as soon as the child is able to record the information, rather than as the situations occur. Assuming a finite number of situations and that the child has perfect memory, there remain two problems with this approach. First, the parent is constantly learning and revising his own knowledge about each of these situations. The later he passes down his knowledge, the higher-quality the advice. In this tradition, children born to inexperienced parents suffer, compared to children born as the parents grow more mature, and they suffer far more than they would if the parents were allowed to revise their early advice. Second, when there are a lot of possible situations, the cost of communicating and remembering them is high, and most of the information will never be of use to the child. This corresponds to *inherit at state creation* and the number of "life's situations" correspond to the size of the input alphabet.

4 Computing the Probability Estimate

Once a mixture weighting function $W(s)$ and an inheritance evaluation time have been decided upon, how do we express these decisions in an on-line estimation of the probability of a sequence? In on-line probability estimation, we must compute the sum of the likelihoods $P_e(a_i|\hat{s}_i)$ for each a_i in $a_1 a_2 \cdots a_n$, where $\hat{s}_i$ is a state that is specially *selected* as the starting node for the recursive mixture computation using the states excited by $a_1 a_2 \cdots a_{i-1}$. (State selection is the topic of the companion paper [Bun97].) For the present discussion, assume that $\hat{s}_i$ is the maximum-order excited state at time i). If we are performing (arithmetic) coding or decoding, we must also compute the sum of the conditional probabilities, $P_e(b|\hat{s}_i)$, of each b preceding a_i in the (arbitrarily) ordered list of events $b|\hat{s}_i$.

Recall that computing $P_e(b|\hat{s}_i)$ requires access to the ancestor likelihood $P_e(b|\mathbf{suffix}(\hat{s}_i))$. Now, for inheritance evaluation times other than *at every visit*, by definition, for each event $b|\hat{s}_i$, we will not recompute the ancestor likelihood $P_e(b|\mathbf{suffix}(\hat{s}_i))$ for every b that precedes a_i in $\hat{s}_i$'s (the selected state's) event list. Nonetheless, for all states s it is necessary that $\sum_{a \in A} P_e(a|\mathbf{suffix}(s)) \leq 1$, regardless of when the individual $P_e(a|\mathbf{suffix}(s))$ are computed. Ideally we want that sum to be as close to 1 as possible, otherwise, codespace is wasted. The problem is that we cannot count on the current ancestor likelihoods of already-seen events to equal or exceed the ancestor likelihoods we computed for them earlier, since those events may have been seen arbitrarily many times since their likelihoods were computed. Solutions to this problem generally

- under-estimate ancestor likelihoods of veteran events so that their ancestor likelihoods are guaranteed to be less than what they would be if they were recomputed, and
- over-estimate ancestor likelihoods of novel events as they occur to reclaim the ancestor codespace that is wasted by underestimating ancestor likelihoods of veteran events.

4.1 Exclusion

We can reclaim the codespace wasted by over-estimated novel-event likelihoods by subtracting the proportion of the ancestor codespace that corresponds to veteran events, before computing a novel event's proportion of the ancestor codespace. This is best accomplished with a technique known as *exclusion* (not to be confused with update exclusion), which was developed for PPM [Mof90]. The basic idea is this: when *exclusion* is enabled for the model, only consider the frequencies of event $a|\mathbf{suffix}(s)$ if the higher-order descendant s is currently excited and event $a|s$ has not occurred before. More precisely, redefine $\mathbf{count} : S \rightarrow R$ to be the sum of the event counts for all unexcluded events that have occurred previously following a given state, where a symbol a is defined to be *excluded* at state s if s has an excited child s' such that $\mathbf{count}[a, s', u(s)] > 0$. That is,

$$\mathbf{count}(s) = \sum_{\substack{a \,:\, a \text{ is not excluded} \\ \mathbf{count}[a, s, u(s)] > 0}} \mathbf{count}(a, s).$$

Unless stated otherwise, we shall assume that exclusions are enabled in all computations described from here on.

4.2 Blending's Missing Term

Blending [CW84] evaluates the ancestor likelihood $P_e(a|\mathbf{suffix}(s))$ *before novel event updates*, but at all subsequent occurrences of any string in the set $L(s) \cdot a$. Blending assumes that $P_e(a|\mathbf{suffix}(s)) = 0$, and thereby drops a term of our mixture formula, for events that are not novel. This certainly ensures that the reused ancestor likelihoods for veteran events are less than they would be if they were recomputed from the current frequencies at s's ancestors. The result is that subsequently computed probability estimates of the veteran event $a|s$ are slightly deflated, while exclusions ensure that future estimates of all symbols that have yet to be seen following the a member of the context $L(s)$ will be slightly inflated.

4.3 State Variables for Mixture Computation

In general, the inheritance evaluation time *before novel event updates* makes it difficult to ensure that

$$\sum_{a:\mathbf{count}[a,s,u(s)]>0} P_e(a|\mathbf{suffix}(s)) + \sum_{a:\mathbf{count}[a,s,u(s)]=0} P_e(a|\mathbf{suffix}(s)) \leq 1.$$

However, an alternative is to satisfy the requirement that

$$(1 \Leftrightarrow W(s)) \cdot \left(\sum_{a:\mathbf{count}[a,s,u(s)]>0} P_e(a|\mathbf{suffix}(s)) + \sum_{a:\mathbf{count}[a,s,u(s)]=0} P_e(a|\mathbf{suffix}(s)) \right) \leq (1 \Leftrightarrow W(s)),$$

which can be accomplished less drastically than blending's solution of setting the ancestor likelihood $P_e(a|\mathbf{suffix}(s))$ to zero for any veteran event $a|s$. Instead, we subtract $(1 \Leftrightarrow W(s)) \cdot P_e(a|\mathbf{suffix}(s))$ from the ancestor code space $(1 \Leftrightarrow W(s))$ after the ancestor likelihood is first computed. This is accomplished by implementing the mixture formula using the additional state-variables $\alpha[s]$, which replace $\alpha(s)/d(s)$; and $I[a, s]$, which is required for computing $I(a, s)$ when inheritance evaluation time equals *before novel event updates* (although we use use it for all evaluation times in our cross-product implementation). The other required state variables are the event frequencies, $\mathbf{count}[a, s, 0]$ and $\mathbf{count}[a, s, 1]$, which are required for correctly combining state selection with either type of update exclusion (i.e., *regular update exclusion* or *maximum-order updates*); plus an exclusion vector, *Excluded*$[a]$, which records which symbols were excluded by higher order excited nodes, and which must be reset for each input sequence symbol a_i.

Initially, $\mathbf{count}[a, s, 1] = \mathbf{count}[a, s, 0] = 0$, and $\alpha[s] = z(s)$, where $z(s) = 1$ if weighting function 'A' is used, and $z(s) = 0$ for the mixture variants discussed so far.[3] Then, for each input symbol a_i, after a_i's probability has been computed and its codepoint transmitted, the frequencies corresponding to a_i must be updated at all states excited by $a_1 a_2 \cdots a_{i-1}$ as follows:

[3]For DMC variants, $z(s)$ will be initialized to the frequency of the edge redirected when s was added to the model.

$$
\begin{aligned}
&\forall s : s \in S, a_1 a_2 \cdots a_{i-1} \in L_i(s), \\
\alpha[s] &= \begin{cases} \alpha[s]+1 & \text{if } a|s \text{ is novel and WeightFunction} \neq \text{'A'}, \\ \alpha[s] & \text{otherwise.} \end{cases} \\
\mathbf{count}[a,s,1] &= \begin{cases} \mathbf{count}[a,s,1]+1 & \text{if } s \text{ has no excited children;} \\ \mathbf{count}[a,s,1]+1 & \text{if MaxOrderUpdates = FALSE and} \\ & a|s \text{ or } a|s' \text{ is novel, where } s = \mathbf{suffix}(s'); \\ \mathbf{count}[a,s,1] & \text{otherwise.} \end{cases} \\
\mathbf{count}[a,s,0] &= \mathbf{count}[a,s,0]+1.
\end{aligned}
$$

The value of $I[a,s]$ is determined during probability estimation, for each excited state s that equals the selected state $\hat{s}_i$ or one of its ancestors. In a compressor or decompressor, probability estimation is intertwined with arithmetic coding. The relationship between arithmetic (de)coding and probability estimation that computes mixtures of suffix-tree frequency distributions is described in the recursive procedure *code*: $S \times A \times \{0,1\} \rightarrow [0 \ldots 1)$ given in Figure 1.

The procedure in Figure 1 shows how the set of inheritance times affect the recursive mixture computation when applied to on-line coding. However, the pseudocode is designed for generality without obfuscating the conceptual simplicity: any actual implementation (including ours) must differ to be more computationally efficient and to avoid the instabilities of floating-point arithmetic. One optimization in particular bears mention: with any of the escape mechanisms 'A', 'B', 'C', or 'D', and *inherit before novel event update*, $I[a,s]$ can be permanently subtracted from $\alpha[s]$ if **count**() is redefined to be $\mathbf{count}(a,s) = \mathbf{count}[a,s,u(s)] + k + I[a,s]$.

During compression, *code*($\hat{s}_i, a_i$,1) computes and returns the probability of the currently scanned sequence symbol a_i and incrementally transmits the high-order bits of the decoder's global variable *Codepoint*, which identifies the unique subinterval of $[0 \ldots 1)$ that corresponds to a_i, using an arithmetic coder. During decompression, *code*($\hat{s}_i, \lambda, 0$) incrementally receives the high-order bits of the *Codepoint*, and computes and returns the probability of the symbol, a_i, that corresponds to the unique subinterval of $[0 \ldots 1)$ that contains the value of *Codepoint*. (The probability of a_i equals the width of a_i's subinterval in $[0 \ldots 1)$.) The procedures *arith_renorm_transmit()* and *arith_renorm_receive()* manage the arithmetic coder's and decoder's internal states, respectively, including the decoder's *Codepoint*. The *Codepoint* is incrementally transmitted (high-order bits first) using the subinterval endpoints that are specified at the low end by

$$sum/_{\text{denominator}}(W(s)),$$

and at the high end by

$$(sum + {}_{\text{numerator}}(W(s)) \cdot \mathbf{count}(a_i, s) + I(a_i, s))/_{\text{denominator}}(W(s))$$

if event $a_i|s$ is not novel, or

$$(sum + \alpha[s])/_{\text{denominator}}(W(s))$$

if $a_i|s$ is novel.

```
procedure code(s ∈ S, a ∈ A ∪ {λ},  Coding ∈ {0,1})
        r, x, sum, I_s ∈ ℜ;   b ∈ A;
        sum ← 0.0;   r ← numerator(W(s))/count(s);   I_s ← Σ_{c:Excluded[c]} I[c,s];
        repeat  b ← symbol of next unexcluded event in s's event list;
                if Exclusion then Excluded[b] ← True endif
                if Inherit_Time = At_Every_Event_Visit then
                        ∀p ∈ ancestors(s) ∪ {s},  I[b,p] ← P_e(b|suffix(p)) · α[p]
                endif
                I_s ← I_s + I[b,s];   x ← r · count(b,s) + I[b,s];
                if Coding then
                        if b ≠ a then sum ← sum + x
                        else         arith_renorm_transmit(sum, x, denominator(W(s)))
                        endif
                else
                        if sum + x < Codepoint · denominator(W(s)) then sum ← sum + x
                        else         a ← b;   output a;
                                     arith_renorm_receive(sum, x, denominator(W(s)), Codepoint)
                        endif
                endif
        until b = a or s's event list is exhausted;
        if b ≠ a then
                if Coding then arith_renorm_transmit(sum, α[s] − I_s, denominator(W(s)));
                else     arith_renorm_receive(sum, α[s] − I_s, denominator(W(s)), Codepoint);
                endif
                Insert novel event a|s into s's event list;
                I[a,s] ← (α[s] − I_s) · code(a, suffix(s));   x ← I[a,s]
        else if Exclusion then ∀b|s ∈ s's event list, Excluded[b] ← False endif
        return(x/denominator(W(s)))
end procedure
```

Figure 1: On-line (de)coding of event $a = a_i$ using recursive *mixture* with *inheritance*.

Table 1: How average compression performance on the Calgary Corpus as a whole is affected by varying mixture inheritance times and mixture weight functions, in models with and without (percolating) state selection.

Inherit Time	*A*9X*	*B*9X*	*C*9X*	*D*9X*	*A*9XS*	*B*9XS*	*C*9XS*	*D*9XS*
M_0	2.965	4.695	2.825	2.767	2.602	2.925	2.522	2.508
M_2	**2.631**	**2.505**	2.365	2.306	**2.386**	2.674	2.227	2.206
M_3	2.767	2.688	2.329	**2.281**	2.475	2.559	**2.197**	**2.191**
M_5	2.851	2.520	**2.300**	2.302	2.482	**2.419**	2.203	2.238

Table 2: The percent improvement of models using update exclusion over the same model variants without update exclusion.

Inherit Time	*A*9X*	*B*9X*	*C*9X*	*D*9X*	*A*9XS*	*B*9XS*	*C*9XS*	*D*9XS*
M_0	1.8	-6.5	1.1	2.9	2.0	0.2	6.9	2.1
M_2	5.5	3.2	2.2	5.9	5.7	3.7	3.2	5.4
M_3	5.9	-0.8	2.9	7.5	5.5	2.7	3.8	6.5
M_5	5.8	5.3	4.3	10.2	1.4	6.2	4.3	6.4

5 Empirical Comparisons

In this section we address the following questions:

1. Which mixtures perform best?
2. How do the various mixture weighting formulae and inheritance times interact?
3. Is the effectiveness of update exclusion affected by the mixture with which it is combined?

Table 1 shows the relative effectiveness of most combinations of mixture weighting functions and inheritance evaluation times. Inheritance time M_1 (*inherit at state creation*) was omitted because it exhibits impractical space consumption for models with 256-character input alphabets. As expected from past experience with PPM, weight functions C and D produce the best results.

Table 2 is a study on the value of using update exclusion, especially in models using state selection. This study is important because the largest cost of correctly implementing state selection with lazily evaluated model refinements, or of combining approximate state selection with mixtures, is keeping two counts for every transition: an update-excluded count and a full-update count. For example, since the percolating state selector S_3 improves an order-9 model with mixtures DM_3i_0 and update exclusion X_1 by about 8.3%, and update exclusion improves an order-9 model with mixtures DM_3i_0 and percolating state selector S_3 by about 6.5%, it probably would not be worth the trouble to implement state selection if doing so would require disposing of update exclusion. Basically, the later the inheritance evaluation time, the more update exclusions improve performance.

Table 3 shows how the better performing mixtures, highlighted in **bold** in Figure 1, perform on individual files.

Table 3: Compression performance for the best inheritance times given each weighting mechanism

	size	*A*9X*	*B*9X*	*C*9X*	*D*9X*	*A*9XS*	*B*9XS*	*C*9XS*	*D*9XS*
File	*(bytes)*	M_2	M_2	M_5	M_3	M_2	M_5	M_3	M_3
bib	111,261	2.089	2.129	1.895	1.884	1.910	2.025	1.809	**1.794**
book1	768,771	2.576	2.438	2.391	2.393	2.223	2.238	**2.194**	2.198
book2	610,856	2.175	2.152	1.987	1.996	1.938	1.990	1.876	**1.871**
geo	102,400	6.081	4.460	4.842	4.724	4.939	**4.432**	4.455	4.511
news	377,109	2.666	2.623	2.385	2.363	2.465	2.546	2.300	**2.298**
obj1	21,504	4.555	3.973	3.785	3.737	4.343	3.956	**3.654**	3.704
obj2	246,814	2.755	2.699	2.368	2.340	2.657	2.657	2.317	**2.302**
paper1	53,161	2.579	2.652	2.347	2.340	2.397	2.579	2.268	**2.250**
paper2	82,199	2.552	2.527	2.354	2.347	2.299	2.412	2.232	**2.219**
pic	513,216	0.889	0.804	0.817	0.807	0.820	**0.781**	0.790	0.797
progc	39,611	2.669	2.728	2.373	2.365	2.514	2.650	2.305	**2.301**
progl	71,646	1.777	1.962	1.600	1.590	1.677	1.878	1.576	**1.548**
progp	49,379	1.878	2.055	1.652	1.659	1.737	1.940	1.603	**1.566**
trans	93,695	1.598	1.864	1.396	1.389	1.483	1.786	1.377	**1.320**
Avg.		2.631	2.505	2.300	2.281	2.386	2.419	2.197	**2.191**

Our controlled component-wise experiments with the Calgary Corpus show that the best-performing mixtures outperform models that assume a uniform prior frequency distribution at every state by about 23% in models without state selection, and by about 16% in models that use state selection, on the Calgary Corpus. We also demonstrated how mixtures interact with update exclusion, which improves performance as much as 6 $\Leftrightarrow$10%. Broadly speaking, the later the inheritance evaluation time, the greater the impact of update exclusion.

Not surprisingly, the overall best-performing weighting formulas (escape mechanisms) were C and D. Regardless of the other parameters, mixtures that *inherit before novel event updates* consistently outperform blending by about 1%, when used with the competitively performing mixture weights C and D or with state selection.

6 Conclusion

We must always compute some sort of *mixture* when computing on-line probability estimates with suffix-tree FSMs, and a recursive mixture is completely defined in terms of its recursive weighting function $W(s)$, and its *inheritance evaluation time*. For example, PPM's *blending* is a forgetful type of mixture that lazily evaluates its inheritances as novel events occur, while DMC's "cloning" [CH87] produces a mixture that evaluates its inheritances when new states are added, but which also subtracts the inherited frequency from the parent distribution (see Chapter 6 of [Bun96] for details). The weighting functions and inheritance evaluation times are independent of each other and of whether inheritances are subtracted from parent distributions; thus *mixtures* generalize both PPM's *blending* and the quantitative aspects of DMC's *cloning*.

References

[BCW90] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Advanced Reference Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[Bun96] S. Bunton. *On-Line Stochastic Processes in Data Compression.* PhD thesis, University of Washington, December 1996.

[Bun97] S. Bunton. A percolating state selector for suffix-tree context models. In *Proceedings Data Compression Conference.* IEEE Computer Society Press, March 1997.

[CH87] G. V. Cormack and R. N. S. Horspool. Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550, 1987.

[CW84] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[How93] P. G. Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Brown University, 1993.

[Mof90] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[TR93] J. Teuhola and T. Raita. Application of a finite-state model to text compression. *The Computer Journal*, 36(7):607–614, 1993.

[WB91] I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

[Wil91] R. N. Williams. *Adaptive Data Compression.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991.