A Percolating State Selector for Suffix-Tree Context Models[1]

Suzanne Bunton

Technical Report UW-CSE-97-02-06

Department of Computer Science and Engineering
University of Washington

Abstract

This paper introduces into practice and empirically evaluates a set of techniques for performing information-theoretic *state selection* that have been developing in asymptotic results for over a decade. State selection, which actually implements the selection of an entire *model* from among a set of competing models, is performed at least trivially by all of the suffix-tree FSMs used for on-line probability estimation. The set of state-selection techniques presented here combines orthogonally with the other sets of design options covered in the companion papers, "A Generalization and Improvement to PPM's *Blending*," and, "An Executable Taxonomy of On-Line Modeling Algorithms," written by this author. The main results of this paper are:

- a novel dynamic programming solution that does not resort to the suboptimal hill-climbing or global order bounds that are used in other techniques,
- the successful combination of information-theoretic state selection and em mixtures, which include PPM's *blending*, and
- the first published survey and empirical comparison of state selection techniques.

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

[1]A version of this paper appears in Proceedings of the DCC, March 1997. This report contains minor corrections to the DCC97 version.

A Percolating State Selector for Suffix-Tree Context Models

Suzanne Bunton
The University of Washington

This paper introduces into practice and empirically evaluates a set of techniques for information-theoretic *state selection* that have been developing in asymptotic results (e.g., [Ris83, Ris86, WLZ92]) for over a decade. State selection, which actually implements the selection of an entire *model* from among a set of competing models, is performed at least trivially by all of the suffix-tree FSMs used for on-line probability estimation. The set of state-selection techniques presented here combines orthogonally with the other sets of design options catalogued in the companion paper [Bun97a].

1 Stochastic Complexity and On-Line Modeling

The *stochastic complexity* of a string is the length of its optimal off-line encoding, that is, its Minimum Description Length, or MDL [Ris89]. A string's MDL is the sum of the lengths of an encoding of a model plus the encoding of the string with respect to that model such that the total encoding length is minimal over all possible models within an assumed model class. Here we consider the problem of computing the input sequence's MDL on-line, by assuming for each input symbol that the set of possible FSM models is represented by the set of nested subtrees of the suffix-tree FSM. Our goal is to code each input symbol with the model that assigns the lowest stochastic complexity to the already processed portion of the input. At any point in the input sequence, each nested FSM is in a particular state. The set of current states of the nested FSMs is the set of *excited* suffix-tree states. By selecting one of the excited states, we are in fact selecting an entire FSM with that state as its current state.

Note that stochastic complexity assigns a coding penalty to each model state. The penalty is a lower bound on the number of bits required to encode that state. Most other treatments of the state-selection approach to on-line modeling (e.g., [Ris83, WLZ92]) require that a refinement (e.g., the children or deeper descendants) of a state be selected in preference to that state if the performance of that refinement improves the performance of the model frontier containing the original state by an amount exceeding the cost of encoding the states comprising the refinement. However, for the following reasons, we believe that there should be no coding penalty for selecting a refinement to a given state during *on-line* modeling:

- In on-line modeling, the model is not explicitly coded: the deterministic algorithm of the encoder is emulated by the decoder to deduce the state that was selected for coding without any side-information about the actual model.
- In on-line modeling, the coding penalty is incorporated into the inaccurate probability estimates from early in the sequence [CW84]. Thus, refinement coding penalties are incorporated into the records of past performance based on those estimates.
- Automated experiments with various parameterizations of our executable taxonomy, which evaluate the effect of loose constant lower bounds on the off-line

coding cost of model refinements, indicate that positive coding penalties always degrade on-line performance.

Therefore, for each input symbol a_i, we shall simply select the excited state that represents the model that has *performed* the best on the input sequence $a_1 a_2 \cdots a_{i-1}$.

2 A Performance Metric for States

Thus we need to associate a *performance metric* with each state to use for minimizing the codelength of the entire sequence $a_1 a_2 \cdots a_n$. The optimal codelength equals

$$\sum_i - \log P(a_i | a_1 a_2 \cdots a_{i-1}),$$

and we model all past occurrences of all suffixes of $a_1 a_2 \cdots a_{i-1}$ using the currently *excited* states $\{s : a_1 a_2 \cdots a_{i-1} \in L(s)\}$. We therefore wish to minimize

$$- \log P_{\text{estimated}}(a_i | \{s : a_1 a_2 \cdots a_{i-1} \in L(s)\}),$$

for all a_i, where $L(s)$ is the set of strings that cause the nested FSM represented by the suffix-tree to enter state s or any state in the subtree rooted by s.

A performance metric that will help accomplish this goal is maintained as follows: For each state s we maintain a counter called $D[s]$ that accumulates the codelengths that state s would have assigned to the symbols that were currently scanned whenever it has been excited. That is, for each input a_i, at all s such that $a_1 a_2 \cdots a_{i-1} \in A^* L(s)$, we increment $D[s]$ by $- \log P_e(a_i | s)$. We also keep track of $\sharp s$, the number of times that s has been excited since it was added to the state set. Then, we measure the performance of each state s by its expected codelength, $D[s]/\sharp s$. From here on we shall loosely refer to each state's expected codelength $D[s]$ as its "MDL." Now we are prepared for selecting the best-performing model represented by the excited states.

3 Basic Approaches to State Selection

There are three basic approaches to information-theoretic state selection. Each approach can be viewed as selecting a *complete frontier* of the subtree rooted at s_0, where a frontier of state s's subtree T consists of the leaves of some subtree of T rooted at s, and a frontier is *complete* if it consists of s or of complete frontiers of all of s's children.

1. Top-down from s_0, select the first state s whose children's combined expected codelengths fail to improve s's expected codelength.
2. Bottom-up from the maximum order excited state, select the excited child of the first state s whose children's combined expected codelengths improve s's expected codelength.
3. Top-down, select the minimum order excited state for which no complete frontier of its subtree improves its expected codelength.

The first approach was introduced in [Ris83], using entropies instead of MDLs, and then in [Ris86] using MDLs. As pointed out in [WLZ92], this technique systematically under-estimates the local order of the model; it is a hill-climbing technique that can get stuck in local minima. The second approach is introduced here as an obvious complement to the top-down hill-climbing approach, to complete the taxonomy. It systematically over-estimates local order. Both methods are most efficiently implemented using a single MDL counter $D'[s]$ at each state s, which records the difference between the per-symbol codelength assigned by the state s and the codelength assigned by the currently excited child of s. That is, given that $D'[s] = 0 \; \forall s \in S$ initially, for every state s such that s has an excited child t at time i, let

$$D'[s] \leftarrow D'[s] - \log(P_e(a_i|s)) + \log(P_e(a_i|t)).$$

Both hill-climbing methods approximate the third approach. The reason that the hill-climbing approaches are suboptimal is that there may be a complete frontier below any given state's children that reduces the state's expected codelength, even though the children themselves do not. The authors of [WLZ92] present a formal, asymptotically convergent solution to the third approach that requires an order bound. In the next subsection, we describe our own solution, which requires no order bound and which allows efficient implementation. But first we will explain certain semantic considerations involving context partitions and frequency updates.

4 Model Semantics: Competing Context Partitions

State selection in a suffix-tree FSM implements the selection of an entire partition on the set of possible conditioning contexts, where the partition element associated with each state is the set of strings that will cause the FSM to enter that state or any state in its subtree. We restrict ourselves to the selection of complete frontiers because only complete frontiers impose a complete partition on the set of conditioning contexts. For every possible input history $a_1 a_2 \cdots a_{i-1}$, there must exist a state in the coding model selected at time i whose conditioning context contains $a_1 a_2 \cdots a_{i-1}$.

It is fortunate that a state-selection procedure need only consider the metrics located at the excited states. However, the designer must remember that he or she is actually implementing the selection of an entire conditioning context partition from among a set of competing partitions. This means that a node must always be considered for selection *along with its siblings.* To select, for example, the excited state with the lowest expected codelength, or the minimum-order excited state whose expected codelength is better than that of its excited child (e.g., [Fur91]), is incorrect, not merely suboptimal. This is because the children of a state s (i.e., those nodes whose contexts correspond to minimal extensions of the state's context) may have better performance than the state s, even while the currently excited child of s has worse performance than s does.

Frequency updates affect the conditioning context partition imposed by the state set. Therefore, the choice of update mechanism must be carefully considered in combination with state selection. At first glance, only *full updates*, which increment the frequency distribution at every excited state (and therefore at every simulated FSM model), seem appropriate for models with state-selection. However, there are

```
**procedure** $\textit{Select}(s \in S)$
        $\textit{selected} \in S \cup \{\textit{null}\}$;
        **if** $s \neq s_0$ **then**
                $\textit{selected} \leftarrow \textit{Select}(\textbf{suffix}(s))$;
                **if** $\textit{selected}$ **then** **return** $\textit{selected}$ **endif**
        **endif**
        **if** $D[s] \leq F[s]$ **then** **return** $s$ **else** **return** $\textit{null}$ **endif**
**end procedure**


**procedure** $\textit{Percolate\_MDLs}(s \in S,\, a \in A)$
        $\textit{old\_D},\ \textit{old\_F} \in \Re$;
        $\textit{old\_F} \leftarrow F[s]$;
        $F[s] \leftarrow F[s] - \log(P_e(a|s, \textit{update\_exclusion} = \textit{true}))$;
        **while** $s$ **do**
                $\textit{old\_D} \leftarrow D[s]$;
                $D[s] \leftarrow D[s] - \log(P_e(a|s, \textit{update\_exclusion} = \textit{false}))$;
                **if** $\textbf{suffix}(s)$ **then**
                        $\textit{diff} \in \Re$;
                        $\textit{diff} \leftarrow \textit{min}\{D[s], F[s]\} - \textit{min}\{\textit{old\_F}, \textit{old\_D}\}$;
                        $\textit{old\_F} \leftarrow F[\textbf{suffix}(s)]$;
                        $F[\textbf{suffix}(s)] \leftarrow F[\textbf{suffix}(s)] - \textit{diff}$
                **endif**
                $s \leftarrow \textbf{suffix}(s)$
        **end while**
**end procedure**
```

Figure 1: A State Selection Mechanism with Percolating Updates

two reasons to combine state-selection with *update exclusion*, which increments frequencies at only the highest-order excited states. Update exclusion improves performance of mixtures (which we propose to combine with state selection), and, as we argue in Section 5.1, update exclusion correctly handles the *incomplete* frontiers that result from lazily evaluating refinements to suffix-tree states. With mixtures, the probability estimate is defined recursively in terms of ancestor nodes. State selection does not cause us to assume that the children of these lower-order nodes do not exist, therefore update-excluded frequencies should be used to compute the lower-order terms of the probability estimate, even when state selection is employed. Regardless of whether the model uses mixtures or constructs incomplete frontiers, update-excluded frequencies must not be used to compute the probability estimate at the selected state, since the act of selecting a state assumes that the descendants of the selected state do not exist. Thus, disabling update exclusion at the currently selected state is required when it is enabled globally for the modeling algorithm. The dual update mechanism introduced in [Bun96], Chapter 5, provides this ability by maintaining both update-excluded and full-update frequencies at every state.

5 A Percolating State-Selection Mechanism

Here we present a dynamic-programming solution to the problem of finding the best-performing model frontier without resorting to hill-climbing or an order-bound. In simple terms, our solution recursively "percolates" the performance of each subtree's best frontier up to its root's ancestor. Figure 1 gives the two procedures, *Percolate_MDLs*: $S \times A \rightarrow \{\}$ and *Select*: $S \rightarrow S \cup \{null\}$ that implement the two principal steps. These procedures require two MDL accumulators at each actual state s, $D[s]$ and $F[s]$, which respectively contain the state's locally accumulated MDL, and the accumulated MDL of the best complete frontier in the subtree rooted by the state.

Select(s') follows suffix pointers from the current maximum-order excited state s' to the root s_0, and then searches top-down for the first excited state s such that $D[s]$ is no greater than $F[s]$. For each excited state s, *Percolate*(s, a_i) recomputes, bottom-up, the local description length, $D[s]$, and $F[s]$, the description length of the best-performing frontier in the s's subtree. The procedures in Figure 1 implement the following bottom-up recomputation of $F[s]$ for each state s, starting from the maximum-order excited state s', by resetting $F[\mathbf{suffix}(s)]$ to

$$F[\mathbf{suffix}(s)] - \min\{D[s], F[s]\} + \min\{D[s] - \log P_e(a_i|s), F[s] - \log P_e(a_i|s_j)\},$$

where s_j is the excited state located on the best performing frontier of the subtree rooted by s.

For on-line suffix-tree modeling algorithms in general, calls to *Select* and *Percolate_MDLs* fit into the sequence of calls to the other routines as follows. At the beginning of processing the sequence $a_1 a_2 \cdots a_n$, the model consists of the (excited) state s_0, and its parent, s_{-1}, where $F[s_0] = D[s_0] = F[s_{-1}] = D[s_{-1}] = 0$. Before each subsequent symbol a_i is processed by the probability estimation routines, *Select* is called to select the best excited state $\hat{s}$. Then, $P(a_i|a_1 a_2 \cdots a_{i-1})$ is estimated as $P_e(a_i|\hat{s})$. After a_i has been processed by the probability estimation routines, the maximum-order excited state s' is tested for eligibility to be extended by new states. If any new states s are added to the model, they are added top-down as suffix-linked descendants of s', before *Percolate_MDLs*() is called. The MDL counters at new states are initially zero, that is, $F[s] = D[s] = 0$, and the pointer to s' is set to the maximum-order novel state. Then, the MDL counters are updated by executing *Percolate_MDLs*(s', a_i). Finally, the event frequencies are updated at all excited states before advancing to the next scanned sequence symbol a_{i+1} and setting s' to the maximum-order state that is excited by $a_1 a_2 \cdots a_i$.

5.1 Model Semantics: Incomplete Frontiers

Most implementations of data compression algorithms, including the methods studied in this work, add children states *on demand* instead of all at once. Thus, few states in the suffix-tree will posess all possible children. Here, we explain how to use update-excluded frequencies during MDL updates to ensure that the MDLs keep track of the performance of models corresponding to *complete* context partitions.

In an on-line suffix-tree FSM in which a state s does not posess all possible children, the refinement to s's context partition element $L(s)$ that is represented by s's children is incomplete because

$$L(s) - \bigcup_{t:\mathbf{suffix}(t)=s} L(t) \neq \{\}.$$

To complete s's context partition element, we maintain a "shadow child" of s that maintains a next-symbol frequency distribution that is conditioned by

$$L(s) - \bigcup_{t:\mathbf{suffix}(t)=s} L(t).$$

As explained in [Bun96], Chapter 5, this is the same frequency distribution that is formed by the update-excluded frequency counts $\mathbf{count}[-, s, 1]$ if maximum-order updates are globally enabled for the modeling algorithm. Alternatively, it is approximated very closely by the update-excluded frequency counts $\mathbf{count}[-, s, 1]$ if regular update exclusion is globally enabled instead. *Therefore, update-excluded frequencies, in addition to full-update frequencies, are required to correctly implement state selection in context models that lazily refine their context partitions, independently of whether update-excluded frequencies are used when estimating the coding distribution.*

Whenever no state is selected from among the currently excited states, the "shadow child" of the maximum order excited state s', is the selected state, and the local order estimate equals the order of s' plus one. To implement the selection of a complete frontier, the probability estimator should use the update-excluded frequencies conditioned by s' when computing the coding distribution, and when updating the MDL $F[s']$ of the best-performing frontier below that state. That is, during MDL updates, increment $F[s']$ by $-\log P_e(a_i|s', \mathit{update\_exclusion} = \mathit{true})$, which equals the shadow child's codelength, and update $D[s']$ by the maximum order excited state's codelength, $-\log P_e(a_i|s', \mathit{update\_exclusion} = \mathit{false})$. The quantity $P_e(a_i|s', \mathit{update\_exclusion} = u)$ equals a weighted sum of the *inheritance* $I[a_i|s']$, and the maximum likelihood of a_i given the frequencies $\mathbf{count}[b, s', u]$, $\forall b \in A$, as computed in [Bun96], Chapter 6.

6 State Selection with Mixtures

In general, an on-line modeling algorithm that uses convergent state selection will ignore the high-order descendants of a given node until the combined estimated probability distributions of those descendants have better performance, or lower entropy, than the frequency distribution at the given node. A simple explanation of why blending and other mixtures work so well in on-line modeling algorithms is that these techniques enable a given state's probability estimate to converge to the characteristics of the input data sooner. Accelerating this convergence is essential to a state's usefulness if it has high Markov order. Even for large input sequences, high-order states are invariably starved for data. In practice, mixtures accelerate the convergence at particular *states*. That is, mixtures lower the expected entropy of their estimated probability distributions.

Thus, the combination of mixtures with state selection will accelerate the convergence of *models*. That is, higher-order states will be selected sooner than with state selection alone. The algorithm resulting from this combination should produce a model that performs well for both short and long sequences.

7 Comparative Performance

In this section we seek to answer the following questions:

1. Does state selection improve performance in practice?
2. How do the different state selection mechanisms perform relative to each other?
3. What are good threshold values for the information-theoretic techniques?
4. Do the techniques perform predictably for models of different orders?
5. Which improves performance more: state selection, mixtures, or both?

The suite of experiments that we ran to answer these questions covered the cross product of the following sets of parameters:

State Selection Techniques in $\{S_0, S_2, S_3, S_5, S_6\}$, where

S_0 denotes no state selection.

S_2 is the heuristic used by the original PPM* implementation: select the lowest-order excited state that recognizes only one source symbol.

S_3 is our percolating state selector.

S_5 is top-down hill-climbing state selection.

S_6 is bottom-up hill-climbing state selection.

Probability Estimators in $\{AM_0, DM_3X\}$, where

AM_0 denotes the degenerate mixture that produces an identical uniform prior for each state. In the language of [Bun97a], it combines a constant uniform weighting function on the excited states (mixture-weighting formula A), and evaluates their inherited probabilities *at model creation*, denoted by the inheritance evaluation time M_0.

DM_3X describes one of the better-performing mixtures from [Bun97b], combined with update exclusion (X). The mixture-weighting formula (or escape mechanism) is D, while the inheritance evaluation time M_3 equals *inherit before novel event updates.*

FSMX Model Topologies in $\{9^*, 64^*\}$: The model is built using our suffix-tree construction algorithm [Bun96, Bun97c] (thus the *), to save space and time and to therefore make it possible to evaluate higher order models (i.e., orders 9 and 64) at a wide range of state selection thresholds in a reasonable time frame.

State Selection Threshold Numerator v, where v is a member of the set $\{-1024, -512, -256, \ldots, 512, 1024\}$. Note that the actual threshold equals $v/1024$. Thus state selection tests for differences between the expected codelengths of a given excited state and a complete frontier below it that are bounded above by one bit per source symbol. In hill-climbing techniques, the frontier is formed by the children of the state, while the percolating technique uses the complete frontier with the minimal codelength.

All other features of the models were held constant throughout the experiment. The results of the experiment are summarized in Tables 1 and 2. Table 1 compares performance of state selection techniques on a vanilla FSM model with Markov order 64 on the files of the Calgary Corpus. Table 2 compares performance of state selection techniques on an otherwise identical model that also performs one of the better-performing mixtures. At the bottom of each table we summarize the performance on the Corpus for the order 64 models, denoted by the column headers, and for otherwise identical models with Markov order 9. Next to the bits per character figure for each file and model, we give the average selected order of that model. Note that

Table 1: Effect of different state selection techniques on the compression performance and average selected order of a order-64 FSMX model without blending or Mixtures.

File	A*64M_0S_0		A*64M_0S_2		A*64$M_0S_6v_0$		A*64$M_0S_5v_{-16}$		A*64$M_0S_3v_0$	
		Select		*Select*		*Select*		*Select*		*Select*
	(bpc)	*Order*	(bpc)	*Order*	(bpc)	*Order*	(bpc)	*Order*	(bpc)	*Order*
bib	2.749	12.63	2.558	5.35	2.435	3.99	2.312	3.41	**2.286**	3.32
book1	3.452	8.32	3.374	6.49	2.873	4.28	2.444	3.13	**2.432**	2.85
book2	2.857	10.46	2.715	6.45	2.461	4.63	2.198	3.59	**2.189**	3.44
geo	6.059	4.54	6.040	3.42	5.548	2.12	5.280	2.20	**5.216**	2.01
news	3.302	13.12	3.170	5.88	2.999	4.22	2.822	3.50	**2.794**	3.30
obj1	4.584	12.98	4.526	7.87	4.579	3.18	**4.507**	7.55	4.531	3.37
obj2	2.989	14.33	2.851	5.19	2.885	3.89	2.845	3.79	**2.829**	3.87
paper1	3.208	8.98	3.097	4.80	2.944	3.54	2.756	2.77	**2.748**	2.77
paper2	3.300	8.00	3.204	5.23	2.911	3.62	**2.588**	2.67	2.590	2.59
pic	1.148	47.13	1.052	38.34	0.898	19.67	0.993	35.37	**0.852**	5.19
progc	3.143	9.12	3.013	4.56	3.002	3.40	2.888	3.02	**2.883**	2.98
progl	2.314	19.26	2.098	6.86	2.094	4.38	2.066	4.73	**2.044**	3.85
progp	2.224	20.92	1.975	5.63	2.002	4.61	1.989	3.54	**1.976**	4.19
trans	2.025	25.66	1.775	5.71	1.776	4.29	1.756	3.90	**1.727**	3.95
Average	3.097		2.961		2.815		2.675		**2.650**	
Average for order-9 models:										
	3.020		2.946		2.804		2.674		**2.654**	

Table 2: Effect of different state selection techniques, on the compression performance and average selected order of an order-64 FSMX model with Update Exclusion (X) and Mixtures (M_3, D).

File	D*64M_0S_0		D*64M_3S_2		D*64$M_3S_6v_0$		D*64$M_3S_5v_{-8}$		D*64$M_3S_3v_0$	
		Select		*Select*		*Select*		*Select*		*Select*
	(bpc)	*Order*	(bpc)	*Order*	(bpc)	*Order*	(bpc)	*Order*		*Order*
bib	2.019	12.63	1.828	5.35	1.816	5.10	1.801	4.19	**1.788**	4.54
book1	2.428	8.32	2.407	6.49	2.332	5.74	**2.205**	4.38	**2.205**	4.44
book2	2.099	10.46	1.970	6.45	1.937	5.95	1.878	4.72	**1.876**	5.09
geo	4.727	4.54	4.756	3.42	4.705	2.97	4.550	2.79	**4.508**	2.28
news	2.452	13.12	2.347	5.88	2.329	5.37	2.301	4.57	**2.292**	4.62
obj1	3.793	12.98	3.766	7.87	3.757	7.39	3.720	7.53	**3.699**	3.29
obj2	2.422	14.33	2.280	5.19	2.288	4.99	2.287	4.59	**2.272**	4.74
paper1	2.411	8.98	2.302	4.80	2.288	4.46	2.256	3.67	**2.249**	3.96
paper2	2.394	8.00	2.326	5.23	2.288	4.75	**2.218**	3.65	2.221	3.92
pic	0.969	47.13	0.849	38.34	0.785	35.62	0.819	35.74	**0.795**	21.13
progc	2.440	9.14	2.304	4.56	2.310	4.28	2.307	3.88	**2.296**	4.08
progl	1.766	19.26	1.530	6.86	1.539	5.67	1.554	5.92	**1.528**	5.30
progp	1.767	20.92	1.504	5.63	1.516	5.46	1.550	4.85	**1.505**	5.39
trans	1.611	25.66	1.306	5.71	1.306	5.50	1.315	4.68	**1.291**	4.88
Average	2.378		2.248		2.228		2.197		**2.180**	
Average for order-9 models:										
	2.281		2.250		2.235		2.202		**2.191**	

the average order of any model using state selection is the average selected order of the corresponding models that use no state selection, minus one.

Now we are prepared to answer the 5 questions posed earlier:

1. All forms of state selection tested improve performance. The higher the Markov order, the greater the improvement. The improvement due to state selection is greater if the model does not use one of the better-performing mixtures, which hedge against order over-estimation by mixing in data from lower-order states.
2. The performance increases of each state selector, relative to the vanilla order-64 model, were: 4.4% for S_2, 9.1% for S_6, 13.6% for S_5, and 14.4% for S_3. The performance increases of each state selector, relative to the order-64 model with mixtures but no state selection, were: 5.5% for S_2, 6.3% for S_6, 7.6% for S_5, and 8.3% for S_3. Since 9 closely estimates the global order of the corpus, the selectors obtained 2-4% less of a performance increase in similar order-9 models.
3. The percolating and bottom-up selectors always perform best with their state selection threshold set to zero. Top-down hill-climbing performs best when its threshold is set to a small negative value that compensates for its tendency to underestimate local order. The ideal value is a function of the true order of the input sequence. For most files of the Calgary Corpus the top-down selector did best with a values between $-8/1024$ and $-32/1024$. However, with this selector a threshold that does best for the low-order files would get worse performance on the higher-order files, and *vice versa.* Since higher-order text files dominate the corpus, slightly negative values gave the best results.
4. The performance ranking of the selectors is consistent for all Markov orders that we have tested, regardless of whether the models use mixtures or not. The percolating state selector S_3 consistently outperforms the other selectors on all files and at all Markov orders that we have tested. However, the top-down hill-climbing selector S_5 can be parameterized to perform nearly as well.
5. Despite the consistent behaviors of the state selectors, *mixtures*, including *blending*, provide about one and a half times the performance increase of the best state selector. The mixture used in this test improved the performance of the order-64 vanilla model by 23.2% and the order-9 vanilla model by 24.5%.

In summary, a good mixture works better than the best state selector, but the combination is better still. The best-performing state selector is our percolating technique, followed closely by top-down hill climbing—but only if it is correctly parameterized. An unbounded-order model combined with the percolating state-election technique satisfies a primary goal of universal on-line modeling: eliminating model parameters that cannot be automatically deduced from the input sequence.

8 Conclusion

Before we began the basis of this work in [Bun96], there had been no published empirical studies of the performance of any information-theoretic state-selection technique, nor was the idea used in any published algorithm that had been implemented. In this paper, we gave an overview of concepts and the existing state-selection techniques from the information-theoretic literature, and then presented a novel mechanism that overcomes the drawbacks of the existing techniques, which resort to order bounds or suboptimal hill climbing. This paper also describes a major component of the orthogonal sets of an executable cross-product taxonomy for on-line suffix-tree models of sequences, that we present in the companion paper [Bun97a]. We used our implementation of that cross product to present controlled experiments that conclusively demonstrate the principle hypothesis of [Bun96]: that the combination of information-theoretic state selection and mixtures is superior to either technique alone.

Acknowledgements

John Cleary suggested the percolating solution to the general problem of informing a tree's root node of the performance (as described by some arbitrary metric) of the leaves of the best subtree rooted by it.

References

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

[Bun97a] S. Bunton. An executable taxonomy of on-line modeling algorithms. In *Proceedings Data Compression Conference.* IEEE Computer Society Press, March 1997.

[Bun97b] S. Bunton. A generalization and improvement to PPM's *blending.* UW-CSE Technical Report UW-CSE-97-01-10, The University of Washington, January 1997.

[Bun97c] S. Bunton. Semantically motivated improvements for PPM variants. *The British Computer Journal, Special Data Compression Issue*, 1997. (invited paper, to appear June 1997).

[CW84] J. G. Cleary and I. H. Witten. A comparison of enumerative and adaptive codes. *IEEE Transactions on Information Theory*, 30(2):306–315, 1984.

[Fur91] G. Furlan. An enhancement to universal modeling algorithm 'context' for real-time applications to image compression. In *IEEE Transactions on Acoustics Speech and Signal Processing*, pages 2777–2780, 1991.

[Ris83] J. J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, 1983.

[Ris86] J. J. Rissanen. Complexity of strings in the class of Markov sources. *IEEE Transactions on Information Theory*, 32(4):526–532, 1986.

[Ris89] J. J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing, Singapore, 1989.

[WLZ92] M. J. Weinberger, A. Lempel, and J. Ziv. A sequential algorithm for the universal coding of finite memory sources. *IEEE Transactions on Information Theory*, 38(3):1002–1014, 1992.