An Executable Taxonomy of On-Line Modeling Algorithms[1]

Suzanne Bunton

Technical Report UW-CSE-97-02-05

Department of Computer Science and Engineering
University of Washington

Abstract

This paper gives an overview of our decomposition of a group of existing and novel on-line sequence modeling algorithms into component parts. Our decomposition, and its implementation, show that these algorithms can be implemented as a cross product of predominantly independent sets. The result is all of the following: a *test bed* for executing controlled experiments with algorithm components, a *framework* that unifies existing techniques and defines novel techniques, and a *taxonomy* for describing on-line sequence modeling algorithms precisely and completely in a way that enables meaningful comparison.

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.

An Executable Taxonomy of On-Line Modeling Algorithms

Suzanne Bunton
The University of Washington

This paper gives an overview of our decomposition of a group of existing and novel on-line sequence modeling algorithms into component parts. Our decomposition, and its implementation, show that these algorithms can be implemented as a cross product of predominantly independent sets. The result is all of the following: a *test bed* for executing controlled experiments with algorithm components, a *framework* that unifies existing techniques and defines novel techniques, and a *taxonomy* for describing on-line sequence modeling algorithms precisely and completely in a way that enables meaningful comparison.

Full descriptions of the algorithms, and definitions of the technical terms used in this overview, are given in Chapters 2–6 of [Bun96], excerpts of which appear as the companion papers [Bun97b, Bun97a].

1 Design Philosophy

The executable cross product and nomenclature described here are used to produce and describe the experimental results of [Bun97b, Bun97a], which evaluate different state selection and probability estimation techniques, respectively. The cross product is also used in [Bun97c], which significantly improves the performance and memory requirements of PPMC [Mof90] and PPM* [CTW95]. The design of our software involved the following steps:

1. Identify a set of basic, independent features that most sequential models share.
2. Transform influential techniques from the literature to a unifying control and data structure that decomposes into the basic features.
3. For each basic feature, define the set of interchangeable options corresponding to existing, abstractly distinct implementations of that feature.
4. For each basic feature, try to improve upon the existing solutions, and add those improvements to the set of options.
5. Implement the cross product of the sets of interchangeable options for each feature. Each element in the cross product should be a viable, working on-line modeling algorithm. The cross product will properly include the original algorithms from the literature, and should reproduce their results exactly.

Because we combine major components (e.g., state selection and mixtures) and several minor components (e.g., update exclusions and inheritance times) for the first time, and because we added novel options to the component sets (e.g., percolating and bottom-up state selection, and new edge-redirection tree structures), the cross product implements many genuinely novel on-line modeling algorithms. Furthermore, it provides complete experimental control for the evaluation of individual model features.

2 The Common Control Structure

The main control loop of the cross-product implementation performs the following computations in sequence for each input symbol, using an adaptive FSM model of the input sequence:

1. Excite the model states whose conditioning context partitions contain the processed input sequence, and make the transition into the next maximum order excited state, initially the root node, and subsequently determined in step 6.
2. Select the coding model, which is represented by one of the excited states.
3. Estimate the probability of the currently scanned symbol using the frequencies at the selected state and possibly its ancestors.
4. Add new descendants to the maximum-order excited state via splitting or redirection of the incoming transition taken, if it is eligible. The maximum-order novel descendant becomes the new maximum order excited state.
5. Add a novel event (i.e., out-transition) corresponding to the currently scanned symbol to each novel excited state or any excited state that "missed." (They will be the highest-order excited states.)
6. Compute the next maximum-order excited state and prepare the next-state transitions into all simulated models. This step also manages the suffix-linked virtual nodes in the string-transition suffix-tree implementation.
7. Update the MDLs at all currently excited states.
8. Compute *before-update* inheritances at excited actual states, if applicable.
9. Update frequencies at excited states.
10. Compute *after-update* inheritances at excited states, if applicable.

3 The Cross Product of Distinguishing Features

The algorithmic variants presented and tested in [Bun97b, Bun97a] are specified in the column headings of the tables using the conventions described below for describing the major features of modeling algorithms: model structure, probability estimation, frequency updates, and state selection. The description below accomplishes four goals:

- It assigns English names to each feature option, to form a language for precisely describing on-line statistical algorithms.
- The symbols accompanying the English names give a terse labeling system that completely describes on-line algorithms.
- It explains our software's command-line options (see Figure 1).
- It synopsizes the modeling concepts presented in [Bun96, Bun97b, Bun97a, Bun97c].

```
Usage: runDMC [-B(atch)-U(decode)-b-e-s-v-o-r-P-y-z-s-w-i-m-K-x-G-D-M-c] file
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Alphabet_Bits:                    (the log of the input alphabet size)
        -b {1, ..., 16}  (Default = 8)
neg_log_EPSILON: (EPSILON = minimum frequency.  Resolution R of floating point
        arithmetic coder (R >= 32) and Max File Size determine Minimum EPSILON.
        -e {1, ..., R} (DEFAULT = 10 allows a Max File Size of 4.19 M symbols).
                       (Max Frequency = Max File Size = 2^{R-(-log_2(EPSILON))}.
State_Selection:
        -s 0 (Default) select no state
        -s 2 select min-order deterministic excited state; else select no state
        -s 3 select excited state on Min-Entropy frontier; else select no state
        -s 5 same as 3, but approx Min-Entropy frontier top-down;
        -s 6 same as 3, but approx Min-Entropy frontier bottom-up;
State_Selection_Threshold:        (small means nodes mature at young age)
        -v <integer> (Default = 0, actual threshold == n/1024)
Minimum_Order:          (max order of nodes added by string event splitting)
        -o -1 Infinite Order      (PPM_Star)
        -o  0 Zero Order + nodes added by edge reclassification (Default, DMC)
        -o  {1, ..., 32767}       (PPM, PPM-DMC hybrid where higher-than-min-order
                                   nodes are added by eligible event reclassification)
Reclassification_Eligibility;     (for last_taken guest in_event)
        -r 0  Never (Default, PPM, nodes are added ONLY by string-event splitting)
        -r 1  FREQUENCY: if freq(curr_node)+ 1.0 - freq(in_edge) > Thresh'd,
        -r 2  MDL: if guest dest'n->parent->MDL- dest'n->parent->extnsMDL > Thresh_other
        -r 3  Destination Order < Min_Order;
Owner_Edge_Protection:     (don't redirect edges more than once);
        -P  (DEFAULT= off)
Reclassification Thresholds:    (small value speeds growth)
        -y <integer> (edge threshold  = v/1024.0, Default = 0.0)
        -z <integer> (other threshold = w/1024.0, Default = 0.25)
Reclassification_Tree_Structure:   (redirect guest events/edges to new classes/states)
        -t 0  one edge to one new state  (DMC)
        -t 1  chain of linked guest_edges to one new state
        -t 2  chain of linked guest_edges to sibling set of new states
        -t 3  chain of linked guest_edges to suffix chain of new states
Mixture_Weights:        P_est(a|n) = #n/(#n+q) * #a/#n + q/(#n+q) * P_est(a|n->parent)
                                where #n = Sum_{b:count(b|n) >= 1}{#b}, and
        -w A  #a = count(a|n); q = Initial_Mass_Of_Parent_Mixture;
        -w B  #a = count(a|n) - 1; q = |{b: count(b|n) >= 1 }|
        -w C  #a = count(a|n); q = |{b: count(b|n) >= 1 }|
        -w D  #a = count(a|n) - 0.5; q = 0.5* |{b: count(b|n) >= 1 }|
Initial_Mass_Of_Inheritance:      (new nodes'initial escape counts before updates)
        -i <integer> Default: = 0.0(do not use 0.0 with -p A)
Mixture_Inheritance_Time:         (when is P_est(a|n->parent) computed relative to n?)
        -m 0  at model creation Uniform Prior
        -m 1  at  node creation   DMC
        -m 2  at novel event prediction,  Blending:
                                              count(a|n) >=1 => P_est(a|n->parent)=0.0
        -m 3  before novel event updates
        -m 4  after novel event updates       (like 3, slightly more aggressive)
        -m 5  at every event visit
Satisfy_Kirchoff:           Subtract inheritance from parent's distribution.
        -K (DEFAULT = off).       In edge's counts will equal out_edge's counts.
Update_Exclusion:                 (which nodes use in_situ freqs, or subtree freqs too)
        -x 0  Off (Default): every state uses counts gathered from its entire subtree
        -x 1  PPM Update Exclusion: Only selected state uses counts from its subtree.
        -x 2  Maximum-Order Update Exclusion
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Figure 1: The command-line options of the executable taxonomy.

3.1 Model Structure and Growth

Suffix-tree model[2] structure is determined by the size of the input alphabet, the initial model, any order bounds, and whether the model is implemented with symbol-labelled transitions or string-labelled transitions (which are described in [Bun96, Bun97c] and denoted with a '*'). Thus, the following features specify model structure:

Alphabet Bits, b: DMC uses a binary input alphabet, with $b = 1$; the other techniques we test use a 256-ary alphabet, with $b = 8$.

Minimum Order: This global bound o guarantees that at time i, for every unique substring w of $a_1 a_2 \cdots a_{i-1}$ such that $|w| \leq o$, there will be a unique state s in the suffix tree with conditioning context string $\textbf{context}(s) = w$. Either the suffix-tree has no global order bound (default, given as '-1') or an order bound is specified simply as a scalar (e.g., '3').

Order-Zero Initial Model, M: Required to exactly emulate DMC and GDMC. Early predictions are negligibly better with an order -1 initial model, the default.

Transition Redirection: Two distinct operations are used to build models: transition *splitting* builds string-transition models, while transition *redirection* builds symbol-transition models. Thus the model structure is additionally determined by how and when symbol-labelled transitions are redirected.

Redirection Criteria: These determine the eligibility of the transitions taken into the currently excited states for redirection.

never, R_0 (or equivalently, $*$): Only add nodes by splitting string transitions. (Implements linear-space PPM and PPM*.)

popularity, R_1: Redirect transition if and only if its count exceeds threshold $y/1024$ and the contribution to its destination's count by other in-edges exceeds threshold $z/1024$. (Implements DMC and GDMC.)

MDL, R_2: Redirect transition if and only if the best-performing frontier below its destination's parent has an expected minimal codelength that is $z/1024$ base e bits per character less than that of its parent.

order, R_3: Redirect transition if and only if the order of its destination is less than the Minimum Order. (Used for full-space impl. of PPM.)

Owner Protection, P: this option limits the redirection criterion to "non-owner edges," that is, edges leaving states with Markov order not less than the order of the destination. Prevents transition re-redirection.

Redirection Tree Structure: These options are logically part of the edge redirection criteria, and can redirect the transitions that were taken into the currently excited states as a group. They are easier to understand if they are left whole, rather than decomposed into single-edge redirection criteria. Models built solely by string-transition splitting are not affected by this option.

one-to-one, T_0: If the edge entering the current maximum-order excited state is eligible, redirect it to one new state. (DMC and GDMC.)

many-to-one, T_1: If the edge entering the current maximum-order excited state is eligible, redirect it and all eligible edges above it to the same new state. (A novel, experimental option.)

[2] Suffix-tree models notably include the FSMX class, and all PPM and DMC variants.

many-to-children, T_2: If the edge entering the current maximum-order excited state is eligible, redirect it and all eligible suffix-linked edges above it to distinct new states that are children of the original destination state, and siblings of each other. (A novel, experimental option.)

many-to-descendants, T_3: If the edge entering the current maximum order excited state is eligible, redirect it and all eligible suffix-linked edges above it to distinct new states that are suffix-linked descendants of the original destination state. (Emulates PPM and WLZ.)

Redirection Thresholds, y and z: for tuning redirection criteria R_1 and R_2.

3.2 Probability Estimation with Mixtures

The companion paper [Bun97a] explains how all probability estimators in the baseline algorithms can be described as recursive mixtures of the frequency distributions at different excited states. The principal features that distinguish any mixture are when, relative to the state's lifetime, each state's inherited distribution is computed and what weight the inheritance is assigned.

Inheritance Evaluation Time: The *inheritance* at a given state s is a frequency distribution constructed from the frequency data present at its ancestors at a given point in time, relative to s's lifetime.

inherit at model creation, M_0: This is the degenerate mixture that corresponds to every state having a uniform frequency distribution initially.

inherit at state creation, M_1: Required by DMC, but too costly for 256-ary alphabets.

inherit at novel event prediction, M_2: This is *blending*, the default mixture for PPM variants. Note that this type of "inheritance" differs from the others in that it is forgotten (i.e., set to zero) after its first use.

inherit before novel event updates, M_3: As a short hand, we call this option M when it will not cause confusion.

inherit at every event visit, M_5: Every time a state becomes excited, recompute the frequency distributions conditioned by its ancestors—they may have changed since last visit.

Initial Mass of Inheritance, i: Let hypothetical variable z be the initial mass of the inheritance, or initial escape count. Then, to correctly emulate PPM variants, z must equal 1.0 for escape method A, and 0.0 otherwise. However, to emulate DMC, each state's initial escape count z must be set to the count on the redirected in-transition.

Mixture Weighting Formula, A, B, C, or D: These options define mixture weights by determining what gets added to the inheritance mass z (which is initialized as described above) at a given excited state after it "misses," and the initial count k of the added event.

A: z remains unchanged, $k = 0.0$;

B: z is incremented by $1.0, k = -1.0$;

C: z is incremented by $1.0, k = 1.0$;

D: z is incremented by $0.5, k = 0.5$.

Zero-Sum Inheritance, K: If this option is enabled, the inheritance given to a state is subtracted from its parent's distribution. Emulates the original DMC.

```
:::::::::::::::::: Options Included to Emulate GDMC Exactly ::::::::::::::::::
Dont_Copy_Event_To_Clones:
        -G  (Default = FALSE, copy to clones if ok by Max_Event_Copy_Depth)
Dont_Update_Freqs_At_Clones:
        -D    (Default = False, update all excited nodes, including new ones)
Order_0_Initial_Model:
        -M (Default = FALSE = use order -1 initial model. No -m0 uniform_prior)
            (when combined with -m1, all nodes will have |alphabet| out_events).
Max_Event_Copy_Depth:
        -c 0  copy predicted event all the way to new leaf (Default)
        -c {1,2,..,255} copy event to c min-order extns of predicting_node
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Figure 2: The special command-line options added to accommodate GDMC.

3.3 Frequency Updates

Chapter 5 of [Bun96] fully describes three update options, how each of them affects the semantics of the model, and how to emulate them simultaneously in a single model that combines non-trivial mixtures and state selection.

Update Exclusion, X: The default for PPM and PPM* variants is no update exclusion for any states. When we need to be more specific we use a longer notation:

full updates, X_0: All excited states get a frequency update.

update exclusion, X_1: Only the excited states that failed to recognize the currently scanned input, and the maximum-order state that recognized it, receive frequency updates.

max-order updates, X_2: Only max-order excited state gets updated.

Special GDMC features: These options, shown in Figure 2, have no abstract rationalization that we know of, but they are necessary for exactly emulating GDMC, and they can be used with any other algorithm in the cross product.

restriction G: Do not copy predicted out-transition to the out-transition lists of newly added states.

restriction D: Do not update the frequencies at newly added states.

copy depth, C: If any excited states "missed", only the C highest-order excited descendants of the maximum order excited state that "hit" receives a copy of the out-transition corresponding to the currently scanned symbol. To emulate GDMC, let $C = 1$.

3.4 The Selection of the Coding Model

The set of state selectors is presented and empirically evaluated in [Bun97b], using a range of state selection thresholds. The percolating state selector below is presented in [Bun97b]. All of our implementations are MDL-based for efficiency, but they could have been implemented using actual entropies of the frequency distributions. We have implicitly assumed throughout our work that MDL-based implementations are equivalent in effect to entropy-based implementations.

```
:::::::::::::::::::::::::::::::: Examples::::::::::::::::::::::::::::::::::::
     DMC: -b1 -s0     -r1 -y2048 -z2048 -t0 -o0  -x2 -wA -i0 -m1 -K       -M -c0
    GDMC: -b8 -s0     -r1 -y0    -z256  -t0 -o0  -x2 -wA -i0 -m3    -G -D -M -c1
     PPM: -b8 -s0     -r3               -t3 -o?  -x1 -wC -i0 -m2             -c0
     PPM: -b8 -s0     -r0                   -o?  -x1 -wC -i0 -m2             -c0
    PPM*: -b8 -s2     -r0                   -o-1 -x0 -wC -i0 -m2             -c0
     WLZ: -b8 -s3 -v? -r3                   -o?  -x0 -wA -i1 -m0             -c0
 Context: -b1 -s5 -v? -r0                   -o-1 -x0 -wA -i1 -m0             -c0
Context2: -b8 -s5 -v? -r0                   -o-1 -x0 -wA -i1 -m0             -c0
BestFSMX: -b8 -s3 -v0 -r0                   -o-1 -x1 -wD -i0 -m3             -c0
 BestDMC: -b8 -s0 -v0 -r1 -y1024 -z2048 -t0 -o0  -x1 -wD -i0 -m3          -M -c0
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Figure 3: Command lines that execute the Markovian baselines, plus others.

State Selectors: These options determine how the coding model is dynamically selected for each input symbol.

none, S_0**:** The default action is to select no state.

heuristic, S_2**:** Select the min-order excited state with only one out-transition ([CTW95]).

percolating, S_3**:** See companion paper [Bun97b].

top-down, S_5**:** An MDL-based implementation of the hill-climbing method used in Rissanen's *Context* algorithm.

bottom-up, S_6**:** A bottom-up complement to S_5.

State-Selection Threshold, v**:** The actual threshold is $v/1024$, and is used to test the difference in expected bits (log base e bits, that is) per character between two models represented by suffix-adjacent states. Higher thresholds cause the algorithm to wait until a state is fairly mature before it is ever used for a prediction.

4 The Command-Line and Baseline Experiments

The command-line "usage" message of our program is listed in Figures 1 and 2 to illustrate how precisely the nomenclature mirrors the actual command-line features. The actual command lines that must be typed to make the cross product emulate the baseline algorithms, ignoring the existance of preprogrammed defaults, are shown in Figure 3. These examples demonstrate the expressive power of our taxonomy—the first six lines plus the final two lines of text completely describe seven different state-of-the-art algorithms[3] and explicitly point out *all* of the abstract differences among them.[4] For each algorithm, the options that are left unspecified do not have any effect on the particular combination of other options.

[3]The seven algorithms are DMC [CH87], GDMC [TR93], PPM [Mof90], PPM* [CTW95], WLZ [WLZ92], plus BestFSMX [Bun97c] and BestDMC, which are the best-performing FSMX and DMC variants tested with this taxonomy so far (see [Bun97a]). PPM is listed twice because there are two very different ways to implement it as a suffix tree, both of which produce identical predictions.

[4]Now, *that's* data compression!

Table 1: The State of the Art in On-Line Statistical Compressors.

File	*Size* (bytes)	LZ78 (compress)	LZ77 (gzip)	DMC	GMDC	PPMC	PPM*
bib	111,261	3.35	2.52	2.28	2.05	2.11	**1.91**
book1	768,771	3.46	3.26	2.51	**2.32**	2.48	2.40
book2	610,856	3.28	2.71	2.25	**2.02**	2.26	**2.02**
geo	102,400	6.08	5.35	**4.77**	5.16	4.78	4.83
news	377,109	3.86	3.07	2.89	2.60	2.65	**2.42**
obj1	21,504	5.23	3.84	4.56	4.40	**3.76**	4.00
obj2	246,814	4.17	2.65	3.06	2.82	2.69	**2.43**
paper1	53,161	3.77	2.80	2.90	2.58	2.48	**2.37**
paper2	82,199	3.51	2.90	2.68	2.45	2.45	**2.36**
pic	513,216	0.97	0.88	0.94	**0.80**	1.09	0.85
progc	39,611	3.87	2.68	2.98	2.67	2.49	**2.40**
progl	71,646	3.03	1.82	2.17	1.83	1.90	**1.67**
progp	49,379	3.11	1.82	2.22	1.90	1.84	**1.62**
trans	93,695	3.27	1.62	2.11	1.73	1.77	**1.45**
Average	224,402	3.64	2.71	2.74	2.52	2.48	**2.34**

The remaining two command lines approximate the original binary-alphabet *Context* algorithm [Ris83] and a 256-ary variant [Ris86]. The emulations are only approximate because the true *Context* model is a non-Markovian FSMX finite state machine, which cannot be represented state-for-state with an FSM that has explicit transitions. However, the true *Context* model is embedded upon the unbounded-order FSMX model constructed by PPM*, since that model contains a state for every substring of the already-processed portion of the input. And since *Context* used top-down hill-climbing state selection, most of the extra states in the Markov FSM will be ignored. If anything, our Markov approximation of *Context* should achieve better performance than the original[5], since the only extra states that will ever be used will be states that tend to improve the performance of the model.

The published performance of on-line stochastic algorithms from the data compression literature that have been implemented are shown in Table 1, along with the performance of two popular Unix compression utilities. The utilities are 'compress,' which is based upon Welch's popular implementation [Wel84] of the Ziv and Lempel's second major string-matching algorithm [ZL78], and 'gzip,' which is based upon Ziv and Lempel's first major string-matching construction [ZL77].

Moffat's 1990 implementation, PPMC [Mof90], of Cleary and Witten's 1984 PPM algorithm [CW84], remained unchallenged until Cleary, *et al.* did away with PPM's order bound to produce PPM* in 1995 [CTW95]. The authors claimed that PPM* outperformed PPMC in their paper. However, PPMC was known to achieve superior compression performance as the order bound increased up to 5 [Mof90], after which its performance starts to decline. In 1993, Howard published a simple change to PPMC's escape mechanism, called PPMD [How93]: add .5 instead of 1.0 to the escape count and scanned event count, whenever a novel event is seen. PPMD gets even better performance than PPMC. Thus, the original PPM* algorithm cannot be called the state of the art until it is shown to perform favorably compared to these

[5]comparable performance baselines for Context are unavailable

Table 2: Cross-Product Baselines of Existing Stochastic Techniques.

File	*Size* (bytes)	DMC	GMDC	PPMC (order 3)	PPM*	PPMC (order 5)	PPMD (order 5)
bib	111,261	2.219	2.045	2.114	1.910	1.915	**1.875**
book1	768,771	2.246	2.319	2.478	2.397	2.338	**2.297**
book2	610,856	2.255	2.021	2.271	2.020	2.004	**1.968**
geo	102,400	4.671	5.157	**4.663**	4.828	4.722	4.712
news	377,109	2.894	2.605	2.648	2.419	2.396	**2.364**
obj1	21,504	4.560	4.403	3.766	4.004	**3.736**	3.737
obj2	246,814	3.064	2.817	2.726	2.434	2.446	**2.421**
paper1	53,161	2.889	2.582	2.482	2.373	2.373	**2.336**
paper2	82,199	2.927	2.451	2.457	2.361	2.358	**2.314**
pic	513,216	0.925	**0.803**	0.823	0.854	0.816	0.808
progc	39,611	2.977	2.666	2.499	2.401	2.411	**2.377**
progl	71,646	2.168	1.826	1.904	**1.671**	1.729	1.693
progp	49,379	2.222	1.905	1.843	**1.624**	1.753	1.719
trans	93,695	2.114	1.734	1.772	**1.447**	1.539	1.495
Average	224,402	2.721	2.524	2.460	2.339	2.324	**2.294**

higher order PPMC and PPMD parameterizations.

Table 2 shows how our emulations of the above statistical techniques perform. The performance is very close to that of the original implementations in Table 1. There are slight differences, however, due to our use of floating point frequencies rather than integers in PPM and PPM* and the frequency scaling that is used in PPM, PPM*, DMC, and GDMC, whenever a frequency at a state exceeds some constant maximum value. Our floating-point arithmetic coder enabled us to dispense with frequency scaling. While we were establishing these baselines, we experimented with our implementation of the existing PPMC and PPMD technologies. And, in spite of its (unbounded) longer conditioning contexts, PPM* clearly does not outperform PPM.

5 Conclusion

Our taxonomical approach to describing and defining novel on-line modeling algorithms constitutes a break with tradition in practical data compression research. Researchers in the past have emphasized (sometimes artificial) differences among their algorithms with memorable acronyms and typically treated existing techniques as "black boxes" (thus often ignoring meaningful parameters) when comparing them to their own constructions. However, quite often the abstract differences among apparently distinct approaches can be expressed in a single technique as different values of parameters that have simple connotations.

References

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

[Bun97a] 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.

[Bun97b] S. Bunton. A percolating state selector for suffix-tree context models. In *Proceedings Data Compression Conference*. IEEE Computer Society Press, March 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).

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

[CTW95] J. G. Cleary, W. J. Teahan, and I. H. Witten. Unbounded length contexts for PPM. In *Proceedings Data Compression Conference*, March 1995.

[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.

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

[Ris86] J. J. Rissanen. An image compression system. In *Proceedings HILCOM 86*, 1986.

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

[Wel84] T. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[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.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, September 1978.