Probability Estimation in Arithmetic and Adaptive-Huffman Entropy Coders

Donald L. Duttweiler, *Fellow, IEEE,* and Christodoulos Chamzas, *Senior Member, IEEE*

***Abstract*— Entropy coders, such as Huffman and arithmetic coders, achieve compression by exploiting nonuniformity in the probabilities under which a random variable to be coded takes on its possible values. Practical realizations generally require running adaptive estimates of these probabilities. An analysis of the relationship between estimation quality and the resulting coding efficiency suggests a particular scheme, dubbed scaled-count, for obtaining such estimates. It can optimally balance estimation accuracy against a need for rapid response to changing underlying statistics. When the symbols being coded are from a binary alphabet, simple hardware and software implementations requiring almost no computation are possible.**

A scaled-count adaptive probability estimator of the type described in this paper is used in the arithmetic coder of the JBIG and JPEG image coding standards.

I. Introduction

ENTROPY coders achieve compression by exploiting nonuniformity in the probabilities under which a signal to be compressed takes on its allowed values. Practical implementations generally require adaptive, on-line estimation of these probabilities either because sufficient statistical knowledge of the signal is lacking or because its statistics are time varying. How best to make such estimates is a question of much practical importance.

We became interested in this problem from our involvement with two standards groups. The joint bilevel image group (JBIG) is under the auspices of both CCITT and ISO and is chartered to develop a standard for progressively coding bilevel (two-tone or black-white) images [1]. The joint photographic experts group (JPEG) is under the same auspices and is chartered to develop a standard for coding photographic (grey-scale or color) images [2], [3]. Both of these groups have finished their work and specifications are available [4], [5].

The JBIG algorithm and the JPEG algorithm (in some of its parameterizations) use a common arithmetic coder [6]–[8]. Although our work was strongly focused on these two image coding applications for arithmetic coders, it is clear that much of what was eventually learned has application beyond just image coding and even for forms of entropy coding other than arithmetic, such as adaptive Huffman.

In the next section of this paper, we review adaptive entropy coding and develop some useful analytical relationships between the quality of probability estimation and the resulting coding inefficiency. The analysis strongly suggests a particular structure for adaptive probability estimation. This structure, which is dubbed scaled-count, makes it possible to achieve any desired balance between the steady-state quality of probability estimation and the speed of response to changing underlying statistics. Section III presents this structure and addresses implementation issues. In some environments (those of JBIG and JPEG among them), both software and hardware implementation become extremely simple, requiring almost no computation.

Manuscript received May 31, 1992; revised November 26, 1993. The assosciate editor coordinating the review of this paper and approving it for publication was Prof. Michel Barlaud.

D. L. Duttweiler is with AT&T Bell Laboratories, Holmdel, NJ 07760 USA.
C. Chamzas is with Democritus University of Thrace, Xanthi, Greece.

IEEE Log Number 9408197.

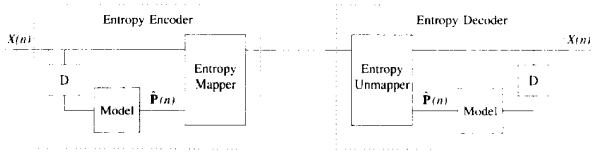

Fig. 1. General model for entropy encoding and decoding.

II. Entropy Coding Review and Analysis

A. General Framework

Let $X(0), X(1), \ldots$ be a random process taking on at each time n one value from a finite alphabet of possible values. Since the particular values taken on will just serve as labels in what follows, they can, without loss of any generality, be taken as $0, 1, \ldots, K-1$, where K denotes the alphabet size.

Fig. 1 shows a general model for one-symbol-at-a-time entropy encoding and decoding. The function of the block labeled "Model" is to provide a vector[1]

$$\hat{\mathbf{P}}(n) = [\hat{P}_0(n), \ldots, \hat{P}_{K-1}(n)]^T \tag{1}$$

of estimates $\hat{P}_k(n)$ of the probability $X(n)$ will take on the value k. It may use any of the observed values $X(0), \ldots, X(n-1)$ in making that estimate. The unit delay block labeled "D" emphasizes that $X(n)$ itself may not be used in forming $\hat{\mathbf{P}}(n)$. If it were to be used, the entropy decoder could not generate a tracking $\hat{\mathbf{P}}(n)$.

Ideally, the block labeled "entropy mapper" places

$$\log_2(1/\hat{P}_k(n)) \tag{2}$$

bits of information on its output stream whenever a symbol $X(n)$ having an instance value k is coded. If the true probabilities $P_k(n)$ of $X(n)$ taking on its various values k are highly

[1] Throughout this paper, vectors are shown in bold-face type. Vector components are separated by commas and grouped by square brackets. A superscript T denotes vector transposition.

nonuniform with some symbols much more likely than others, then the average of the number of bits the mapper uses to code a symbol can be well under $\log_2 K$, and there is substantial advantage to be gained with entropy coding.

Entropy coders are always lossless. The sequence $X(0)$, $X(1), \ldots$ is recovered exactly by the decoder. This is generally the type of coding desired if the sequence $X(0), X(1), \ldots$ represents a text file in a computer system. It is also common for the coding of bilevel images (facsimile) to be done losslessly. In coding other types of images and in coding audio, some coding distortion is often allowed. Even when coding of this sort is being done, however, lossless coding often becomes a part of a larger structure in which some intermediate values, which are invertible to the original signal only with distortion, are themselves coded losslessly. The "baseline" JPEG system provides one example of this.

The name "entropy mapper" is somewhat awkward and is nonstandard terminology. The same is true even more so for the decoding counterpart labeled "entropy unmapper." The reason for coining these terms here is to make it possible to carefully distinguish these mapping functions from the total entropy encoding and decoding functions. The term "entropy encoder" will be reserved for the combination of a probability estimator and an entropy mapper. "Entropy decoder" will always mean the combination of a probability estimator and an entropy unmapper.

The well-known Huffman scheme [9] can attain the ideal of (2) whenever all the $\hat{P}_k(n)$ happen to equal inverse powers of two but, in general, falls somewhat short. The arithmetic mapper used in arithmetic coders does not suffer from this "breakage" problem and achieves the ideal.

When the alphabet size K is large, Huffman mapping often can come acceptably close to the ideal and can be a viable choice for entropy mapping. With binary alphabets, it cannot. The breakage problem is unacceptably severe in the interesting situation where one of the symbol probabilities is small and the other is almost one. If the natural alphabet is binary and it is desired to use Huffman mapping, the standard way to solve the breakage problem is to recast the original problem into an equivalent one with a larger alphabet by grouping consecutive symbols $X(n)$ together. With an arithmetic mapper there is no need to resort to an artifice like this, and indeed, there are significant implementation advantages with the binary alphabet.

Conceptually, an arithmetic coder maps the sequence $X(0), X(1), \ldots$ to a real number on the unit interval [0.0, 1.0). What is transmitted in lieu of the sequence $X(0), X(1), \ldots$ is a binary encoding of this real number. The particular real number to be transmitted is determined recursively as shown in Fig. 2 for an example with a binary alphabet and $X(0), X(1), \ldots$ beginning with the particular instance values 0, 1, 0.

In general, at any given time, there exists a "current coding interval." Initially, it is the unit interval [0.0, 1.0). At time n, it is divided into K subintervals with the kth having a size proportional to $\hat{P}_k(n)$. The current coding interval at time $n+1$ is that subinterval associated with the instance $k = X(n)$ actually occurring.

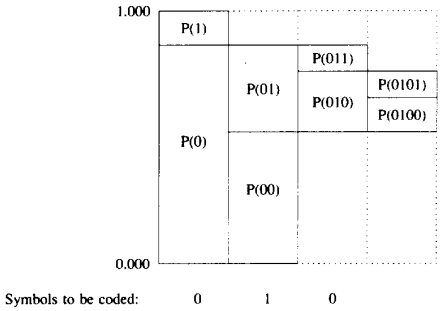

Fig. 2. Interval subdivision example with a binary alphabet.

The idea of coding by recursive interval subdivision is an old idea that is arguably an "obvious" and "immediate" application of the usual equations and theory developed in information theory texts. Abramson [10] credits Elias with having conceived it soon after Shannon's seminal paper [11] on information theory. Nonetheless, arithmetic coding was initially no more that an intellectual curiosity because a straightforward application of the recursive-interval-subdivision idea requires infinite precision arithmetic. What has changed recently is that it has been realized that it is possible to implement recursive interval subdivision with finite precision arithmetic and in a way allowing the "pipelining" of the encoder and decoder [12]–[14]. With a pipelined encoder and decoder, the encoder can begin to put out the bits of the compressed data stream before seeing the entire (possibly infinite) input sequence $X(0), X(1), \ldots$, and the decoder can begin to put out decoded symbols before seeing the entire (possibly infinite) sequence of compressed bits. The existence of finite-precision implementations that can be pipelined is crucial for the practical application of arithmetic coders, but detail about how these properties are achieved is inconsequential and unnecessary for the purposes of this paper. Some additional discussion of arithmetic coding appears later in Section III-F, but it is only to the minimal depth needed to illustrate the points central to this paper. Readers interested in more detail on arithmetic coding are referred to [7] and [8].

B. What is Desired From the Model?

For notational convenience, let $X[i\colon j]$ denote the string $X[i], \ldots, X[j]$. The block labeled "Model" in Fig. 1 is free to use any of the $X[0\colon n-1]$ in forming its estimate $\hat{\mathbf{P}}(n)$. Let

$$\Psi(n) = \{i\colon X(i) \text{ is used in forming } \hat{\mathbf{P}}(n)\} \tag{3}$$

denote the set of time indices of the symbols actually used in forming $\hat{\mathbf{P}}(n)$.

The conditional entropy

$$H(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) = \sum_{k=0}^{K-1} P_k(n) \log_2(1/\hat{P}_k(n)) \tag{4}$$

is the average number of bits that will be used to code $X(n)$ given $X(i), i \in \Psi(n)$. In this equation

$$P_k(n) = \Pr\{X(n) = k \mid X(i), i \in \Psi(n)\} \tag{5}$$

is the true conditional probability of an occurrence of symbol k given the particular pattern $X(i), i \in \Psi(n)$ of past symbols that has occurred. The conditional expected bit count provided by (4) is minimized if

$$\hat{\mathbf{P}}(n) = \mathbf{P}(n). \tag{6}$$

Therefore, not surprisingly, one thing desired from the model for efficient coding is accurate estimates $\hat{\mathbf{P}}(n)$.

More is needed, however. A significant entropy coding advantage is only achievable if the probability distribution $\mathbf{P}(n)$ is nonuniform with one or possibly a few of the symbols taken on by $X(n)$ much more likely than the others. Thus, the other requirement for good modeling is a conditioning set $\Psi(n)$ allowing good prediction so that at least as an average over past patterns $X(i), i \in \Psi(n)$, the probability estimate $\hat{\mathbf{P}}(n)$ is highly nonuniform with one (or, less desirably, a few) symbols predicted to occur with high likelihood.

The unconditional entropy

$$\begin{aligned} &H(\hat{\mathbf{P}}(n) \mid \Psi(n)) \\ &\quad = E[H(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n))] \\ &\quad = \int \Pr\{X(i), i \in \Psi(n)\} H(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) d\mathbf{x} \end{aligned} \tag{7}$$

is the expected bit count to code symbol n using the probability estimate $\hat{\mathbf{P}}(n)$, depending on past symbols $X(i), i \in \Psi(n)$. If $\hat{\mathbf{P}}(n) = \mathbf{P}(n)$ so that the coding is the best possible with the conditioning indices $\Psi(n)$, the expected bit count for coding $X(n)$ is

$$\begin{aligned} H(\Psi(n)) &= E[H(\mathbf{P}_{(n)} \mid X(i), i \in \Psi(n))] \\ &= E\left[\sum_{k=0}^{K-1} P_k(n) \log_2(1/P_k(n))\right]. \end{aligned} \tag{8}$$

It can be shown (see, for example, pp. 105–111 of [10] or almost any discussion on equivocation in an information theory text) that if

$$\Psi_1(n) \subset \Psi_2(n) \tag{9}$$

then

$$H(\Psi_1(n)) \geq H(\Psi_2(n)). \tag{10}$$

Hence, the best model sets

$$\Psi(n) = [0, n-1] \tag{11}$$

to use all the past information and then makes the probability estimate $\hat{\mathbf{P}}(n) = \mathbf{P}$.

From a theoretical standpoint, the discussion thus far says all there is to say about modeling. From a practical standpoint, it says nothing. For most real-world applications, statistical knowledge about the source is usually nowhere near sufficient to begin to calculate

$$\hat{P}_k(n) = P_k(n) = \Pr\{X(n) = k \mid X(i), i \in \Psi(n)\} \tag{12}$$

and even if it were, the calculation would in all likelihood be hopelessly complex.

C. Approximating Excess Bit Count

Before specializing the structure of Fig. 1 to something allowing practical realization, it is useful to proceed a little further within the general framework and derive a broadly applicable approximation to excess bit count.

The conditional entropy difference

$$\begin{aligned} &D(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) \\ &\quad = H(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) - H(\mathbf{P}_{(n)} \mid X(i), i \in \Psi(n)) \\ &\quad = \sum_{k=0}^{K-1} P_k(n) \log_2(P_k(n)/\hat{P}_k(n)) \end{aligned} \tag{13}$$

is the expected excess bit count due to error in probability estimation when $X(i), i \in \Psi(n)$, has been observed. If we assume that the estimate $\hat{P}_k(n)$ is at least close to $P_k(n)$ so that $P_k(n)/\hat{P}_k(n)$ is close to one and the approximation

$$\ln(x) \approx x - 1 \tag{14}$$

with "ln" denoting the natural logarithm is valid, then (13) becomes

$$\begin{aligned} &D(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) \\ &\quad \approx \frac{1}{\ln 2} \sum_{k=0}^{K-1} P_k(n) \left(\frac{P_k(n)}{\hat{P}_k(n)} - 1\right) \\ &\quad = \frac{1}{\ln 2} \sum_{k=0}^{K-1} (P_k(n) - \hat{P}_k(n)) \frac{1}{1 + \frac{\hat{P}_k(n) - P_k(n)}{P_k(n)}}. \end{aligned} \tag{15}$$

The further approximation

$$\frac{1}{1+\varepsilon} \approx 1 - \varepsilon \tag{16}$$

for small ε, and an assumption that $\hat{P}_k(n)$ is sane in the sense that

$$\sum_{k=0}^{K-1} \hat{P}_k(n) = 1 \tag{17}$$

leads to

$$D(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) \approx \frac{1}{\ln 2} \sum_{k=0}^{K-1} \frac{(\hat{P}_k(n) - P_k(n))^2}{P_k(n)}. \tag{18}$$

Let

$$Q(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) = \frac{D(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n))}{H(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n))} \tag{19}$$

denote the fractional (or normalized) excess bit count due to estimation error, that is, the expected coding inefficiency due to estimation error when $X(i), i \in \Psi(n)$ has been observed. A simple formula for this coding inefficiency $Q(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n))$ can be derived under the assumption that one symbol, say, $k_{\max}(n)$, is highly likely to occur, and all the others are unlikely. In the case of a binary alphabet, this will always be true whenever there is anything to be gained from entropy coding. With a multisymbol alphabet, it is a

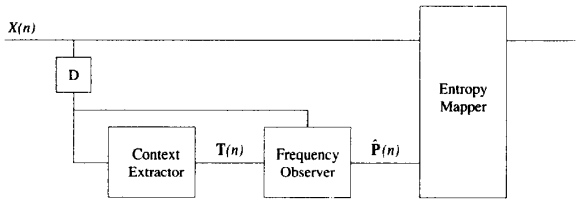

Fig. 3. Adaptive probability estimation for entropy encoding and decoding.

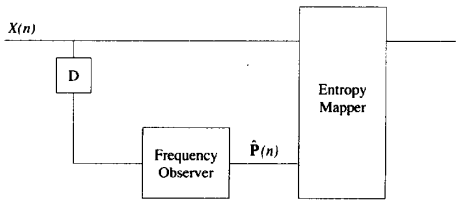

Fig. 4. Simplified adaptive probability estimation with only one context.

somewhat restrictive assumption since entropy coding would be advantageous if a few (as opposed to just one) of the symbols were likely and all the others nonlikely. However, additional structure in multisymbol applications will often justify this assumption.

Assuming the symbol $k_{\max}(n)$ is highly likely and all the others unlikely

$$\begin{aligned} &H(\mathbf{P}_{(n)} \mid X(i), i \in \Psi(n)) \\ &\quad \approx (1 - P_{\max}(n)) \log_2(1/(1 - P_{\max}(n))) \\ &\quad = \frac{1}{\ln 2} \Lambda(1 - P_{\max}(n)) \end{aligned} \tag{20}$$

where

$$P_{\max}(n) = P_{k_{\max}(n)}(n) \tag{21}$$

is the probability of the dominating symbol, and for convenience

$$\Lambda(q) = q \ln(1/q). \tag{22}$$

In words, (20) approximates the entropy $H(\mathbf{P}_{(n)} \mid X(i), i \in \Psi(n))$ by the information in the fact that the expected symbol does not occur times the probability that it does occur. The true entropy also adds in the probability of the expected symbol times its information and the entropy needed to distinguish exactly which of the unlikely events did occur, but these two components to the total entropy are comparatively much smaller.

Using (20) in (19) finally gives

$$\begin{aligned} &Q(\hat{\mathbf{P}}(n) \mid X(i), i \in \Psi(n)) \\ &\quad \approx \frac{1}{\Lambda(1 - P_{\max}(n))} \sum_{k=0}^{K-1} \frac{(\hat{P}_k(n) - P_k(n))^2}{P_k(n)}. \end{aligned} \tag{23}$$

D. Adaptive Probability Estimation

Fig. 3 shows an entropy encoding structure that is much more restrictive than that of Fig. 1, but it has the important advantage of not requiring any *a priori* statistical knowledge about the source for the sequence $X(0), X(1), \ldots$. The decoding counterpart, which is similar, is not shown.

With this adaptive entropy encoder, the conditioning index set $\Psi(n)$ has the special form

$$\Psi(n) = \{i\colon i = n - j \text{ with } j \in \Omega\} \tag{24}$$

where Ω is a set of positive integers. In other words, the condition set $\Psi(n)$ at all times n is always past symbols $X(n)$ at particular offsets relative to the current symbol.[2] The conditioning set Ω is often referred to as a template because it specifies a pattern of past symbols to use in predicting the current one. It is important to note that the template is fixed[3] and not dependent on n. In image coding, the template is usually chosen as a few pixels that are near and causally (in a raster scan sense) related to a given pixel to be coded [16]. In text coding, the template is usually one or two of the preceding characters [7].

The "context extractor" outputs a unique integer $T(n)$ for each possible instance of conditioning values $\{X(n - i), i \in \Omega\}$. For example, if $\Omega = \{1, 2\}$, one possible way to define the context $T(n)$ is by

$$T(n) = X(n - 1)K^0 + X(n - 2)K^1. \tag{25}$$

The exact manner in which particular instances of conditioning values are mapped to an integer is inconsequential as the context $T(n)$ just serves as a labeling in what follows.

The "frequency observer" notes the context $T(n)$ and, based on this, makes a probability estimate $\hat{\mathbf{P}}(n)$. An obvious way in which it might do this adaptively is by maintaining counts of how often each particular instance k of $X(n)$ has occurred when the context has been $T(n)$. Estimated probabilities are roughly (but preferably not exactly, as will be discussed at length later) proportional to occurrence frequencies.

Such a strategy implicitly assumes at least "local" ergodicity so that averages over past time are good predictors of ensemble averages. The qualifier "local" is added because an advantage of considerable practical importance for running (that is, on-line or adaptive) probability estimation like this is an ability to track slowly varying input statistics. This, too, will be the subject of much further discussion.

Context is a powerful idea. It is essential for efficient coding to use it in the JBIG and JPEG environments as well as the text coding environment. In spite of the power and utility of the context idea, the remainder of the analysis in this paper assumes that there is only one context. The reason for doing an analysis under this unrealistic condition is simply that the extension of all results back to the more interesting multicontext environment is trivial. Carrying the necessary notation for context throughout the analysis adds nothing but uneeded clutter. Assuming a single context application, the model of Fig. 3 reduces to that of Fig. 4.

[2] For small n, $\Psi(n)$ may reference nonexistent symbols. There are many reasonable ways of patching this up, and we will ignore this uninteresting detail here.

[3] In the JBIG application, it was found to be advantageous to infrequently allow Ω to change [15], but this kind of detail is best ignored here.

III. Scaled-Count Probability Estimation

A. Bayesian Estimation

Assume the sequence $X(0), X(1), \ldots$ is a sequence of independent, identically distributed, random variables each taking on the value k with probability R_k. If the vector

$$\mathbf{R} = [R_0, \ldots R_{K-1}]^T \tag{26}$$

of symbol probabilities is itself a random variable with *a priori* probability density $f_0(\mathbf{r})$, then the *a posteriori* probability density $f_n(\mathbf{r})$ of the random variable $\mathbf{R}$ given an obseration of $X(0), \ldots, X(n-1)$ is computable by Bayes formula. In particular,

$$\begin{aligned} f_n(\mathbf{r}) &= \Pr\{\mathbf{R} \mid X[0\colon n-1]\} \\ &= \frac{\Pr\{\mathbf{R}, X[0\colon n-1]\}}{\Pr\{X[0\colon n-1]\}} \\ &= \frac{\Pr\{X[0\colon n-1] \mid \mathbf{r}\} f_0(\mathbf{r})}{\int \Pr\{X[0\colon n-1] \mid \mathbf{r}\} f_0(\mathbf{r}) d\mathbf{r}}. \end{aligned} \tag{27}$$

The expected value

$$\hat{\mathbf{R}} = \int \mathbf{r} f_n(\mathbf{r}) d\mathbf{r} \tag{28}$$

is the Bayesian estimate of $\mathbf{R}$.

Analytical evaluation of these equations is possible for some particular *a priori* densities $f_0(\mathbf{r})$. One that is well known is the unform density

$$f_0(\mathbf{r}) = \delta\left(1 - \sum_{k=0}^{K-1} r_k\right) \tag{29}$$

where the Dirac delta function $\delta(\cdot)$ simply forces the probabilities to add to one and lowers the effective dimensionality of the probability density from K to $K-1$. With a uniform *a priori* density

$$\hat{\mathbf{R}} = \frac{\mathbf{C}(n) + 1}{n + K} \tag{30}$$

and

$$f_n(\mathbf{r}) = \alpha(n) r_0^{C_0(n)} \cdots r_{k_1}^{C_{k-1}(n)} \delta\left(1 - \sum_{k=0}^{K-1} r_k\right) \tag{31}$$

where

$$\mathbf{C}(n) = [C_0(n), \ldots, C_{K-1}(n)]^T, \tag{32}$$

$$\mathbf{1} = [1, \ldots, 1]^T, \tag{33}$$

$$\alpha(n) = \frac{\Gamma(n+K)}{\Gamma(C_0(n)+K) \cdots \Gamma(C_{K-1}(n)+K)}, \tag{34}$$

$$\Gamma(x) = \int_0^1 \lambda^{x-1} e^{-\lambda} d\lambda, \; x \geq 0 \tag{35}$$

and the $C_k(n)$ are counts of the number of occurrences of the instance k in $X[0\colon n-1]$. The Gamma function $\Gamma(x)$ is a standard mathematical function that extends the idea of factorial to noninteger numbers. Some of its properties that are useful here are

$$\Gamma(0) = 1 \tag{36}$$

$$\Gamma(x+1) = x\Gamma(x), \; x \geq 0 \tag{37}$$

$$\Gamma(x) = (x-1)!, \; x \geq 1 \text{ and integer.} \tag{38}$$

The estimate $\hat{\mathbf{R}}$ of (30) makes intuitive sense. If n is much larger than K, it becomes simply a ratio of observed frequencies. The "+1" in the numerator and the "$+K$" in the denominator bias early estimates of $\mathbf{R}$ toward equiprobable.

Another *a priori* density that allows analytical evaluation and one that leads to a class of estimates that at least for the JBIG and JPEG applications has been found quite powerful is

$$f_0(\mathbf{r}) = \alpha(0) r_0^{\Delta-1} \cdots r_{K-1}^{\Delta-1} \delta\left(1 - \sum_{k=0}^{K-1} r_k\right) \tag{39}$$

where $\Delta > 0$ is a real number that is a free parameter and the normalizing constant $\alpha(0)$ will be given later. This particular prior has also been described by Zandi and Langdon [17]. If $\Delta = 1$, this density becomes the uniform density already discussed. With $\Delta < 1$, the apriori density $f_0(\mathbf{r})$ tends to favor symbol probability vectors $\mathbf{R}$ near the edges rather than the center of the n-dimensional unit hypercube. Such a density is of interest because the symbol probability vectors it faors are just those for which entropy coding is rewarding. There are of course many other such apriori densities favoring edges and there is no argument for considering one over them other than it happening to lead to analytically tractable calculations.

Under the apriori density 39,

$$\hat{\mathbf{R}}(n) = \frac{\mathbf{C}(n) + \Delta \mathbf{1}}{n + K\Delta} \tag{40}$$

and

$$f_n(\mathbf{r}) = \alpha(n) r_0^{C_0(n)+\Delta-1} \cdots r_{K-1}^{C_{k-1}(n)+\Delta-1} \delta\left(1 - \sum_{k=0}^{K-1} r_k\right) \tag{41}$$

where

$$\alpha(n) = \frac{\Gamma(n+K\Delta)}{\Gamma(C_0(n)+\Delta) \cdots \Gamma(C_{K-1}(n)+\Delta)}. \tag{42}$$

With $\Delta < 1$ (the interesting parameterization), the Bayesian estimate (40) is quicker to jump to conclusions about the underlying probability density. As an example, suppose $K = 2$, and $X(0)$ happens to be 1. Equation (30) (or, equivalently, (40) with $\Delta = 1$) estimates

$$\hat{P}_1(1) = 2/3 \tag{43}$$

whereas with very small Δ, (40) has already concluded that

$$\hat{P}_1(1) \approx 1. \tag{44}$$

Rashness of this sort empirically has been found desirable for the JBIG and JPEG application for which the optimal value of Δ seems to be about 0.4.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.