Chapter 2

Optimal Buffer Constrained Independent Bit Allocation [1]

Contents

2.1 Introduction

In recent years, spurred by an increased interest in efficient digital compression techniques, variable bit rate (VBR) methods for source coding have received renewed attention. In order to exploit the redundancy present in typical non-stationary signals, one has to resort to VBR coding to approach the compression limits set by the

[1]Part of this chapter represents work done jointly with Kannan Ramchandran. For related publications see [71, 73].

source entropy. On the other hand, when the coded bit stream is transmitted over constant bit rate (CBR) channels, buffering is needed to absorb the variations of the instantaneous source bit rate. Even transmission over Asynchronous Transfer Mode (ATM) networks for synchronous applications makes it necessary to buffer the source as excessive delay would produce loss of information.

Studies of buffered variable-length coding of stationary memoryless sources [31] have shown that the probability of eventual buffer overflow is 1 for any buffer size, even when the average rate of the source is matched to the channel rate. Hence, a buffer control strategy is necessary *even for stationary sources.* In the case of non-stationary sources (like typical video sequences), defining efficient buffer control policies for the use of VBR coding techniques in delay limited situations is all the more relevant. Many applications like CD-ROM storage of images and video sequences, windows applications for multimedia workstations, buffer-limited JPEG [112] coding, video broadcast, and MPEG [58] buffer control strategies are *either non real-time or real-time where encoding delay is not an issue.* For these finite-buffer constrained coding applications, computationally expensive methods are admissible if the complexity-performance tradeoff is worthwhile.

It is possible to resort to a large enough buffer size at encoder and decoder in order to absorb source bit rate variations, and thus reduce the significance of the buffer control algorithms. However, this approach may not only be unacceptable because of end–to–end delay restrictions, but also economically unwise even when delay is not an issue, as there may exist "smarter" shorter-buffer solutions that yield the same performance. In particular, the buffer size at the decoder has often to be minimized since it represents a significant cost (e.g. broadcast applications).

This provides the motivation to investigate optimal buffer control strategies for signal sequences in a finite buffered environment. Buffer control algorithms have been considered in the literature in different contexts, especially for video coding. In early studies of video sequence coding [44], as well as in some recent ones [57], the approach was to deterministically map each buffer occupancy level to a fixed coder mode of operation. Some works [119, 74] propose the use of models of the coder behavior to set up coding rate predictions. The control algorithm sets objectives for buffer occupancy and chooses a mode of operation based on these predictions. In other cases ideas from control theory are used to devise the buffer controller [57, 51]. The buffer control is normally designed for a particular encoding scheme and thus scheme-dependent heuristics tend to be introduced [16, 105]. The buffer control problem has been studied in [95, 56] using dynamic programming and a probabilistic model of the coded sequence. Independently of our work, buffered quantization has been studied in [43].

Buffer control schemes typically have the sole objective of preventing overflow, and do not use a global distortion measure over the entire sequence to formulate their strategy. Furthermore they involve zero-encoding delay in their operation, in that they base their algorithm only on the current buffer occupancy level. This leads to the question of how much better one could do if one had access to the entire signal sequence to be coded: i.e. what would be the optimal quantization choice for each element in the sequence *if the entire sequence was known a priori?* In other words, we would be assuming an arbitrarily long encoding delay. The optimality criterion would be a minimum total distortion measure rather than that of only minimum overflow probability, as the former is more meaningful from a source coder's point of view.

Such an optimal solution enables us to quantify the performance tradeoffs involving key design parameters like buffer size or quantization choices and can also serve as a valuable benchmark for assessing the performance of other control algorithms and quantifying their suboptimality. The optimal solution also serves as a starting point for designing less complex algorithms. In particular, we show how a solution that is only slightly suboptimal can be obtained when *only a finite window of the entire sequence is known* at each decision instant, i.e. when a solution with a finite encoding delay is considered. More generally, as will be described, the problem structure lends itself to fast heuristic approximations.

In this chapter (see also [71, 73]) we formalize the generalized problem of buffer-constrained quantization of a sequence of independent blocks and describe how, given a set of quantizers, a finite buffer, and any additive cost measure over the sequence elements, an optimal solution can be found. We show how this problem, one of discrete optimization with constraints, can be construed as a deterministic dynamic programming problem with the Viterbi algorithm used to compute the optimal minimal distortion solution. Note that other work in the literature has considered optimization of the buffer control, either assuming a probabilistic model for the input [104], then using queueing methods, or by including the overflow constraint within the quantizer design loop [22]. Here we do not assume any particular distribution for the source and we compute the optimal solution for a discrete, fixed set of admissible quantizers. We draw parallels between this buffer-constrained quantization problem and the simpler budget-constrained unbuffered quantization problem. We then present a recursive Lagrange-multiplier based algorithm that provides a fast nearly-optimal solution with much reduced complexity.

This chapter is organized as follows. In Section 2.2 we define the problem of buffer constrained quantization. The problem is set in an integer programming framework in Section 2.3. Section 2.4 presents the optimal solution as well as fast heuristic approximations. Finally, Section 2.5 shows some applications of the previously described techniques, namely, providing the optimal buffer size for a given application, benchmarking other buffer control algorithms and quantifying statistical multiplexing gain in ATM networks.

2.2 Problem definition

2.2.1 Quantization and cost criterion

Let us define the system under study. The source, as introduced in Chapter 1, consists of a sequence of "blocks" (for example, entire video frames, 8 by 8 pixels image blocks, or individual samples) that arrive at a constant rate at the encoder. The blocks are to be quantized before being entropy coded and sent to the output buffer for transmission (see Figure 2-1).

For a discrete set of admissible quantizers, the problem consists in choosing that optimal sequence of quantizers which minimizes some cost measure, in a *global* sense. Two points have to be emphasized here:

- The set of quantizers is finite. This is an important factor that determines the type of optimization techniques to be used. Also, we are using "quantizers" as synonym for the generic codebooks $\mathcal{C}_i$ of Chapter 1 so that we are not restricting ourselves to a specific type of encoder.

- Minimization of a global cost measure is necessary to obtain good coding quality. A simple algorithm that locally uses the best quality quantizer (in a greedy fashion) usually results in overall suboptimality.

To find the optimal sequence of quantizers, one has to first define a cost measure consistent with the system's design criteria (see Section 1.4.1). For simplicity, we normally use MSE in our experiments but, in general, our approach can be applied whenever the cost is *additive* over all the elements of the sequence, thus making it possible to consider, for instance, MSE weighted by an activity measure or Human

Figure 2-1: Block diagram of the encoding system. The buffer control mechanism determines the quantizer to be used for each input block. Given the set of quantizers, the system can be characterized by the rate, r_{ij}, and distortion, d_{ij}, for each block and quantizer.

Visual System criteria (see for example Section 2.5.2). Note that our additive cost criterion does not exclude the use of additional pre-quantization stages like a Discrete Cosine Transform (DCT). Also, we place no restriction on the quantization scheme to be used, thus permitting, for instance, the choice of Vector Quantization (VQ).

2.2.2 Delay and buffer size

The system under consideration is made up of three elements: the coder/decoder at the transmitting and receiving ends, with their respective buffers, and the transmission channel. In general, although transmission need not be synchronous (for example, video transmission over ATM networks) there exists a constant delay between:

(i) the time at which the n-th block from the sequence has been acquired by the encoder at the input (T_n^i), and

(ii) the time at which the n-th block from the sequence has to be available to the decoder at the output (T_n^o).

This constant delay ΔT (for all n, $T_n^o - T_n^i = \Delta T$) is due to the encoder and the decoder both being attached to synchronous devices. If the processing time at the decoder is negligible, there are four delay components for each block: (i) encoding delay δt_e, (ii) encoder buffer delay δt_{be}, (iii) channel delay δt_c, and (iv) decoder buffer delay δt_{bd}. Clearly, for any block, the total delay through the system is constant, i.e.

$$\delta t_e + \delta t_{be} + \delta t_c + \delta t_{bd} = \Delta T. \tag{2.1}$$

Note that $\delta_e + \delta_{be}$ is the total delay at the encoder and that the two components can be adjusted independently. δ_{be} is determined by the physical buffer size, while δ_e is due to the encoding algorithm.

This scheme applies to both synchronous transmission applications and ATM networks. In the former, the transmission delay (δt_c) is constant while in the latter it is variable. Moreover, except for non real–time decoding applications, such as image retrieval, the total delay constraint is also present in ATM networks, where transmission delay plays a more important role, as excessive delay due to network congestion could lead to packet loss and therefore quality degradation. Refer to [91] for a detailed analysis of the delay constraints in CBR and VBR transmission environments[2].

From the above, we can establish the equivalence, when the channel rate is finite, of a “finite buffer” constraint and a “finite delay” constraint. These constraints are equivalent since some of the blocks could potentially have a delay greater than the maximum permissible delay in the system (ΔT), if infinite buffers were used with a finite channel rate.

Consider a real–time decoding application where the delay between input and output is constant, and the channel has a constant rate (the encoding delay will normally be constant). An input block that encounters a near empty encoding buffer, i.e. low delay at the encoder (δt_{be} small), will necessarily, from Equation (2.1), be delayed at the decoder, since ΔT is constant. Hence the decoder buffer will tend to fill up if the encoder buffer is emptying and vice versa. From this point of view, in order to be efficient, it is clear that the *buffer memories have to be of equal sizes at both ends* as they must each be able to accommodate the maximum combined buffering delay of $\Delta T - \delta t_c - \delta t_e$.

In some applications, the total delay can be large but buffer memories should remain small to reduce the cost of decoders. From the above, any buffer memory used at the encoder would have to be duplicated at each decoder. On the other hand, the additional encoding delay (δ_e) will just contribute to the total delay, without requiring any additional delay at the decoder. In fact, using techniques to be described in Section 2.4, one could optimize the transmission of a sequence based on all the blocks in the sequence (thus using the maximum encoding delay), while choosing *any physical buffer size*. This approach could be attractive for applications where coding is done once and decoding has to be performed many times, e.g. encoding for storage

[2]We use CBR/VBR for both the transmission environment and the encoding algorithm and it should be clear from the context to which we are referring. Thus for instance one can use a VBR encoding algorithm to transmit over a CBR channel (provided a buffer is used at the encoder).

on CD-ROM.

2.2.3 Possible system configurations

Before we formulate the problem it is useful to list all possible system configurations in terms of the characteristics of both encoding and decoding. The key point here is whether either operation takes place in real time. The following classification is summarized in Table 2.1.

A first application is real–time encoding and decoding (e.g. a videoconferencing system) where delay, as was previously noted, is the limiting factor. Delay is especially important in applications involving interaction between the two ends (for instance a delay of more than 100ms would be unacceptable for videoconference transmission), while it might be less crucial in other applications like live broadcast where no interaction occurs. Roughly, the maximum permissible delay can be allocated between encoding and buffering, thus making the choice of buffer size essentially *delay–limited.* If the transmission has variable delay (as in ATM networks), the constraint on the buffer size at the encoder and the decoder will be tighter [91]. Our algorithm will be less suitable for a situation where total delay has to be small. However, if the delay constraint is not too demanding, as would be the case for broadcast, longer encoding delays may be acceptable and our techniques can be used.

If decoding has to be done in real-time but there is no time constraint for the encoding process (as in digital coding for applications such as video on CD-ROM's), delay is no longer the main issue. Taking as an example the CD-ROM device, or applications such as a video server [18], we note that the only contribution to the delay comes from the decoding buffer, since the data is read at a constant rate from the storage device. Furthermore, the absolute delay between data reading and display will be noticeable only at the start of each "play" mode. Hence the total system delay can be seen as a latency time, and the maximum delay constraint will be more flexible. The limiting factor in the choice of buffer size would be the cost of the physical memory, rather than the delay: the problem is *memory–limited*, rather than delay–limited. Also, the encoding delay can be long since encoding is performed just once (while decoding is done many times) and the encoding delay does not affect the memory needed at the decoder.

The dual case, real-time encoding and non real-time decoding, is essentially equivalent. To encode a sequence in real-time and store it in coded format, instead of displaying it in some output device, the limiting factor is the amount of memory to be used in interfacing the coder output with the storage device: delay is not an issue. Finally, non real-time problems can be essentially seen as "static" non buffer-constrained bit allocation problems which have been studied in the literature [101, 117].

2.3 Problem formulation: integer programming

2.3.1 A first formulation of the problem

In the previous section, we have motivated the relevance of a finite buffer coding environment. Now that both the cost and the constraints have been defined, a first formulation of the problem can be given:

Formulation 2.1 *Given a set of quantizers, a sequence of blocks to be quantized, and a finite buffer, which empties at constant rate, select the optimal quantizer for each block so that the total cost measure is minimized and the finite buffer is never in overflow. The blocks are assumed to be coded independently.*

Encoder	Decoder	Constraint	Example
Real-time	Real-time	Interactive: delay	Videoconference
Real-time	Real-time	Non–interactive: memory	Broadcast
Real-time	Non real-time	Memory	Live Recording
Non real-time	Real-time	Memory	Video CD-ROM
Non real-time	Non real-time	No constraints	File transfer

Table 2.1: All possible configurations of encoding/decoding, their implications in terms of delay/memory constraints and some examples in the context of video coding

In this context, independence of quantization means that a choice of quantizer (codebook) for a given block does not affect the choices for the rest of the blocks, so that the available set of codebooks depends only on the block itself, and the distortion is a function only of $\hat{X}_i$ and X_i. This assumption is not met by video encoding schemes such as MPEG [58] since motion compensated prediction is used. Results for this so called dependent case will be given in Chapter 4.

Note that underflow is not included in this formulation. An underflow constraint would be redundant, given that the objective of minimizing distortion would tend to increase the coding bit rate, and thus automatically penalize underflow. When for some sufficiently long run of blocks the maximum rate (among the discrete set of quantizers available for the blocks) is less than the channel rate, strictly speaking there would be no way to avoid underflow. However it would be easy to add an additional "quantizer" to the admissible set for those blocks for which $r_{max} < r$, where r is the channel rate and r_{max} the maximum rate in the quantizer set. The additional quantizer would have rate equal to r and distortion equal to the minimum distortion achievable within the original quantizer set, i.e. $d(r_{max})$.

2.3.2 Continuous vs. discrete optimization

A fundamental characteristic of the above problem is that the set of available quantizers is *finite*. Consider the following example, where N blocks are coded using three quantizers. Figure 2-2(a) shows the operational rate-distortion (R-D) characteristics of each of these blocks and Figure 2-2(b) represents the composite rate-distortion characteristic, that is, all the values of total rate and distortion obtained using combinations of all admissible quantizer choices for each block. We refer the reader to [21, 22] for a detailed treatment of an operational rate-distortion framework.

The convex hull of the set of points[3] in the composite R-D characteristic, as will be seen in Section 2.4.2.1, contains those solutions that are attainable in a "fast" way and are, if they exist, optimal for each rate. Note that not all rates are attainable, because the convex hull has a discrete number of operating points. As an example, in Figure 2-2(b), suppose we had to allocate a budget of R_1 bits among the N blocks. The only solution which would exactly meet our budget (R_1,D_1) is clearly suboptimal, while a convex hull solution (R_2,D_2) would have to settle for a smaller total rate ($R_2 < R_1$). The optimal solution in this case is (R^*,D^*) which is not attainable using convex hull techniques, but would be reached by our optimal algorithm.

[3]A point (r_1, d_1) belongs to the convex hull if there exists no pair of points (r_i, d_i), (r_j, d_j) in the set such that for a real number $0 \leq \alpha \leq 1$ we would have $\alpha r_i + (1-\alpha) r_j < r_1$ and $\alpha d_i + (1-\alpha) d_j < d_1$. We will later see how Lagrangian techniques can be used to find the points that belong to the convex hull.

Figure 2-2: Individual and composite Rate-Distortion characteristics. Constrained and unconstrained optimization. (a) Each block in the sequence has a different Rate-Distortion characteristic. In this example there are three quantizers available to code each block. (b) For a given choice of quantizers for the blocks in the sequence, we can obtain R-D points to form the composite characteristic. No point in the convex hull meets the budget R_1. The optimal solution R^* is not a convex hull solution. (c) R^* is not a feasible solution with the chosen buffer size.

2.3.3 Constrained vs. unconstrained optimization

In the case where there is no buffering constraint, and only a desired total bit budget has to be met, well-known optimal bit allocation techniques have been documented [101, 30, 117]. We will call this situation an *unconstrained* allocation problem, i.e. one where buffers are either infinite or large enough to store the complete sequence. In general, the finite buffered case, or *constrained* allocation, will be different. As each block is coded, it is sent to an output buffer and therefore the number of bits available for certain blocks will be limited by the constraints imposed by the buffer, namely underflow and overflow conditions. As an example, note that the unconstrained solution (R^*, D^*) of Figure 2-2(b) may produce overflow for a certain finite buffer size (see Figure 2-2(c)) and thus is not a solution to the constrained problem, even though $R^* \leq N \cdot r$ and thus blocks are coded on average with a number of bits lower than the channel rate r. This example should motivate the fact that having buffer constraints increases the complexity of the optimization. However it should be noted that buffering constraints may not always render the unconstrained solution infeasible, so that studying the solution to the unconstrained problem may be helpful in solving the constrained one. For instance in Figure 2-2(c), for the same channel rate r, if the buffer size was increased enough to avoid overflow, the unconstrained solution would *also* be the solution to the constrained problem. This fact will be used later on to define fast approximations.

2.3.4 Integer programming formulation

The problem described in the above section is one of constrained optimization over a set of discrete operating points. What makes the solution of this problem complex is the discrete nature of the space of possible solutions. This problem can be described and solved using techniques developed in the field of integer programming [67, 66]. We now present an integer programming formulation, that will be helpful in developing the optimal solution, although alternative formulations can also be proposed. See Figure 2-1 for the notation.

Consider the allocation for N blocks, and suppose there are M quantizers available to code each block, where each "quantizer" represents a choice of codebook as in Chapter 1. Let d_{ij} and r_{ij} be, respectively, the distortion and the number of bits[4] produced when coding block i with quantizer j, and let r be the channel rate per block. Define an admissible solution x as a selection of one quantizer for each block, i.e. a mapping from $\{1, 2, \ldots, N\}$ to $\{1, 2, \ldots, M\}$, $x = \{x(1), x(2), \ldots, x(N)\}$ where each $x(i)$ is the index of one of the M quantizers for block i. Therefore, $(r_{1x(1)}, r_{2x(2)}, \ldots, r_{Nx(N)})$ and $(d_{1x(1)}, d_{2x(2)}, \ldots, d_{Nx(N)})$ are respectively the rate and distortion for each block and a given choice of quantizers, x.

Now define the buffer occupancy at stage i, $B(i)$, for a given admissible solution x. To account for the fact that the buffer occupancy cannot be negative at any stage (i.e. underflow means the buffer occupancy is 0) we use a recursive definition.

Let $B(1) = r_{1x(1)} + B(0)$, $B(2) = \max(B(1) + r_{2x(2)} - r, 0)$ and, in general:

$$B(i) = \max(B(i-1) + r_{ix(i)} - r, 0), \tag{2.2}$$

where the buffer occupancy at each block instant is increased by the coding rate of the current block and decreased by the channel rate. $B(0)$ is the initial buffer state.

Formulation 2.2 *(Integer Programming)*

The problem is to find the mapping x that solves:

[4]Note that in r_{ij} we include the bits of overhead required to specify that quantizer j was used.

$$\min\left(\sum_{i=1}^{N} d_{ix(i)}\right), \tag{2.3}$$

subject to:

$$B(i) \leq B_{max}, \quad \forall i = 1, \ldots, N, \tag{2.4}$$

where B_{max} is the buffer size.

This problem is similar to the one known in the integer programming literature as the generalized assignment problem (GAP)[5]. We will show in what follows how the problem lends itself to efficient optimal and slightly suboptimal solutions.

2.4 Optimal algorithm and fast approximations

Throughout this section, experimental results will be shown in order to compare the different approaches. In the experiments, we use a JPEG–like coding scheme, with the difference from JPEG [112] being that we permit variable quantizer scales for each 8 by 8 block. Our sequence of signal blocks thus consists of 8 by 8 image blocks coded using M different quantization scales in a JPEG coding environment. Note that while the exact values for attainable Signal-to-Noise Ratio[6] (SNR) are source and coder dependent, the results would still hold for a general class of sources and coders. For convenience, we sometimes consider the normalized buffer size, i.e. B_{max}/r, which indicates the number of blocks that can "fit in the buffer" when coded

[5]In the GAP [32], the objective is to minimize a cost using some resources limited by a set of inequality constraints.

[6]We use Peak SNR as defined by:

$$SNR = 10\log_{10}(255^2/MSE)$$

at the average rate or, in other words, the buffer size expressed in average blocks.

2.4.1 Optimal algorithm

2.4.1.1 Dynamic programming solution using the Viterbi algorithm

The problem under consideration can be solved using the special case of deterministic forward dynamic programming (DP) known as the Viterbi algorithm (VA) [110, 34] for which, given the initial conditions, the best solutions leading to each of the possible final states are found. See Section 1.5.4 for an intuitive description of DP. This technique, successfully employed in the decoding of convolutional codes [82], has also been used in the context of trellis coded modulation (TCM) [106] and trellis coded quantization (TCQ) [65]. We can apply the VA because our system, due to the finite buffer, can be in only a finite number of states. The basic principle consists of creating a trellis to represent all the viable allocations at each instant, given the buffer constraints. Each path in the trellis has an associated cost (the total distortion accumulated by coding the successive blocks with the quantizers chosen in the path) and a certain buffer occupancy at each stage.

More formally we define (Referring to Figure 2-3):

- Trellis: the trellis is made of all possible paths that link the initial stage to nodes in the final stage.
- Stage: Each stage corresponds to a block to be coded.
- Node: each node is a pair (i, b), where $i \in 0, \cdots, N$ is the stage number, and $b \in 0, \cdots, (B_{max} - r)$ is the buffer occupancy state.
- Branch: if quantizer j at stage i has R-D characteristics (r_{ij}, d_{ij}) then node $(i-1, b)$ will be linked by a branch of cost weight d_{ij} to node $(i, \max(b + r_{ij} - r, 0))$ provided no overflow occurs, i.e. $\max(b + r_{ij} - r, 0) \leq B_{max} - r$.

Figure 2-3: Trellis diagram to be used for the Viterbi algorithm solution. Each branch corresponds to a quantizer choice for a given block and has an associated cost, while its length along the vertical axis is proportional to the rate. For instance, quantizer 1 at stage i produces a distortion $d_{i,1}$ and requires rate $r_{i,1}$. A path will correspond to a quantizer assignment to all the blocks in the sequence.

Figure 2-4: The problem seen from the VA point of view: (a) The R-D characteristics of the blocks with available quantizers. (b) Equivalent representation. Each of the branches corresponds to the choice of a specific quantizer and has an attached cost. The length of the branch along the vertical axis is proportional to the rate. (c) All possible paths for the three blocks considered. Paths 1 and 2 cannot be used because of overflow. 1 and 3 are, respectively, the minimum and maximum distortion paths.

We can now use the following algorithm to generate the paths in the trellis (refer to Figure 2-4):

Algorithm 2.1 *Algorithm for trellis growth (Viterbi algorithm [110, 34]):*

Step 0 : *Choose an initial node $(0, B_0)$ and a final node (N, B_N), i.e. the initial and final state of the buffer.*

Step 1 : *At each stage i add permissible branches (as determined by the R-D characteristics of block $i+1$) to the end nodes of all surviving paths. At each node, a branch is grown for each of the available quantizers, and the cost of that branch is added to the total accumulated cost of the path arriving to the node in stage $i+1$.*

Step 2 : *Of all the paths arriving at a node in stage $i+1$, the one having the lowest cost is chosen, and the rest are "pruned". This is made possible due to the additivity and independence of the costs of each branch. Note that, by our definition of branches, paths resulting in overflow are not permitted.*

Step 3 : *Increment i and go to* **Step 1**.

In the rest of the chapter this "unrestricted" VA will be used to evaluate the suboptimality of other approaches as well as to study the characteristics of the optimal solution. Now, we explore modifications of the basic algorithm which aim at reducing the complexity and the encoding delay. First, we study the suboptimality incurred when the number of representative buffer states is reduced (see Section 2.4.1.2) or we limit the encoding delay, i.e the number of blocks over which the optimization is done, so that we perform a sliding window optimization (see Section 2.4.1.3). Then, we note the similarity of our problem to an unconstrained optimization problem, and present techniques that exploit it (see Section 2.4.2).

2.4.1.2 Buffer state clustering to reduce complexity in the VA

It is clear that using the VA to find the optimal allocation for a sequence becomes very complex as the dimensions of the trellis, both the number of states per stage (buffer size) and the number of branches (number of quantizers), increase. See Section 2.4.3 for an analysis of the complexity of the algorithms discussed in this chapter.

A way to reduce this complexity is to decrease the number of states in each trellis stage by "clustering" together some of the states. Instead of considering all possible buffer states, only the minimal cost branch in a range of states is chosen and grown in the following iteration. In our experiment, we use Algorithm 2.1 but we cluster together a fixed number of neighboring nodes or states (the number of nodes that are clustered into one is called the "cluster factor"). The clustering operation consists in choosing the best, i.e. the minimal cost, path among those arriving at a set of neighboring nodes, and then discarding all the suboptimal paths. Thus complexity is reduced, as the number of surviving paths is limited. As an example, Figure 2-5 shows how this approach, with a reduction in the number of VA states by a factor as high as 100 at each stage, produces just over 0.1 dB of degradation over the optimal distortion for our test sequence.

2.4.1.3 Use of VA techniques with limited memory

In general, to guarantee global optimality of the VA for a given sequence, one needs to grow the full trellis before allocating bits to any block, i.e. the encoding delay has to be arbitrarily long. However, many actual implementations require a finite delay and we should thus consider alternatives where paths are grown and released based on just a limited number of blocks.

In Figure 2-6, each plot represents surviving paths after applying the unrestricted VA to the blocks considered. Note that, in both cases, all surviving paths share the same allocation for the first few blocks, i.e. all survivors have common initial branches. This can be interpreted as a finite memory characteristic: as the number of blocks grows, the allocation for the first few blocks is not likely to be influenced by

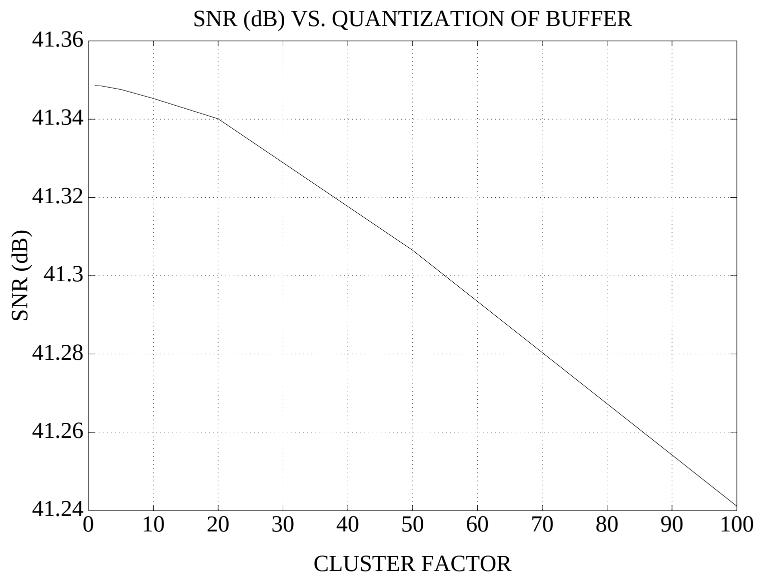


Figure 2-5: SNR vs. cluster factor: The cluster factor indicates how many states or nodes are clustered at each trellis stage. Note that the suboptimality is negligible for cluster factors as high as 100, i.e. when the number of nodes, and thus the complexity, is reduced by a factor of 100.

the allocation for the last ones, which can thus be done independently. Since, for a sufficient number of blocks, all surviving paths have a common root, we can in effect choose the unique path for the first blocks, so that the optimization can be performed in a sliding window fashion. Moreover, note that the sliding window memory, n, is a function of the buffer size (compare Figure 2-6 (a) and (b)). As the buffer size (i.e. the number of possible states) increases, so does the memory of the problem, highlighting a property similar to that of convolutional codes, which typically operate with a finite window of 5 times the constraint length (i.e. the memory of the finite state machine that generates the code [82]).

We now quantify the performance of a finite memory VA. In the previous section, for a length N sequence, we generated a trellis diagram of depth N. The experiment (see Figure 2-7) shows the result of using sliding windows of different sizes, n, for a length N sequence. In order to generate an n-block path, we have to choose a final state at stage $i + n - 1$. This accounts for the non-monotonicity of the SNR with the number of blocks: given that the final state choice for the intermediate paths is arbitrary, a longer encoding delay n does not always guarantee a lower global cost.

Figure 2-7 shows that in the finite memory case, to get within 0.05 dB of the optimal value, one has to consider at least 3 or 4 times the normalized buffer size (i.e. the buffer size divided by the rate per block). It can be seen that the full VA cannot be implemented in real-time, but the results from Figure 2-7 show that even real-time, finite encoding delay implementations of the algorithm produce only slight suboptimality. This is of importance when coding has to be done in real–time but delay can be relatively large (e.g. broadcast). At each instant, the encoder can store as many blocks as needed to find the optimal path, and then release the coded blocks to the buffer with a delay equal to the length of the sliding window. The buffer size at the decoder would be determined by the actual size of the encoding buffer and could thus be kept small.

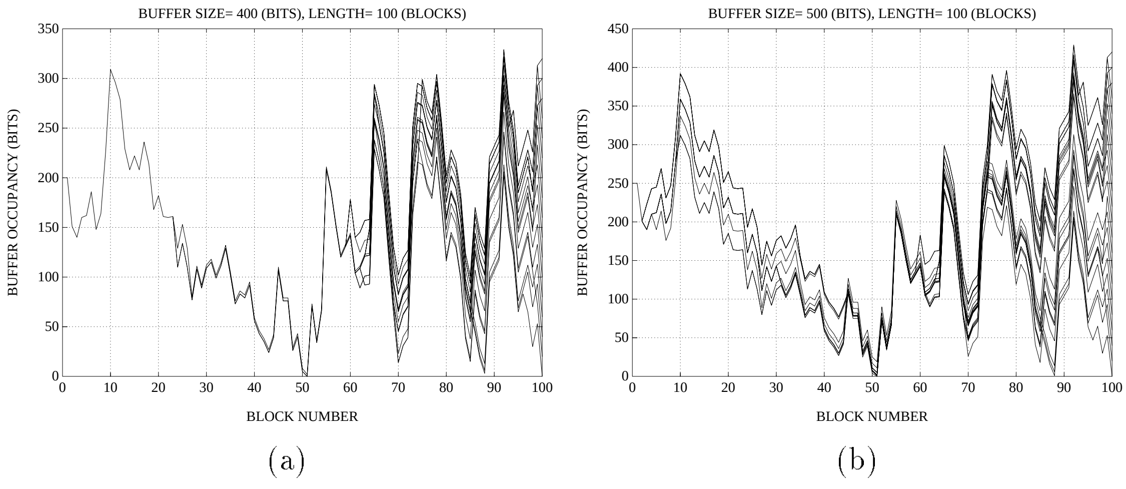


Figure 2-6: Surviving paths using the full VA. The problem has finite memory: for a sufficient length, all surviving paths share the same initial path. B_{max} is the buffer size in bits and L is the number of blocks on which the VA is run. (a) $B_{max} = 400$ and $L = 100$, (b) $B_{max} = 500$ and $L = 100$. Note that as the buffer size is increased the length of the common path decreases.

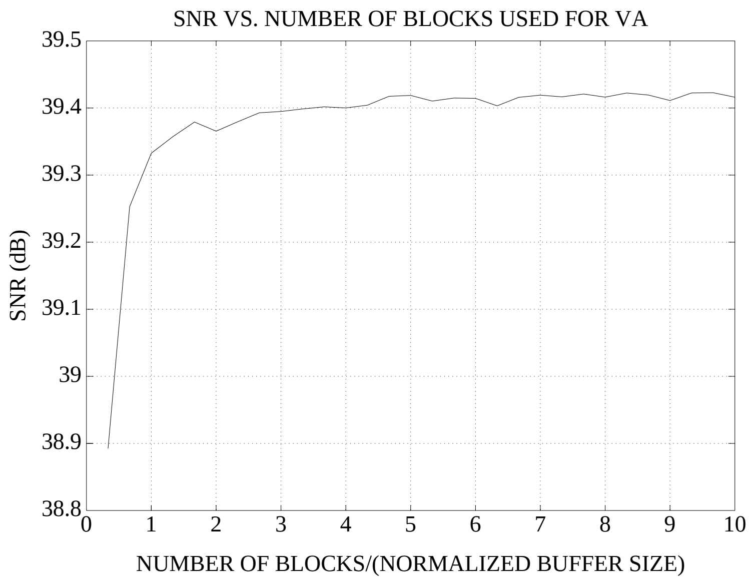


Figure 2-7: SNR vs. number of blocks, n, used to obtain the VA solution. We divide n by the normalized buffer size so that the horizontal axis represents the window of the VA expressed in "number of buffer sizes". In the experiment, a quantizer is chosen for block i based on the best path from i to $i+n$. To select the path, we force it to end at an arbitrary buffer position at stage $i+n$ (normally mid-buffer) thus explaining the non-monotonicity of the resulting function.

2.4.2 Fast approximations

In Section 2.3.3 we noted how our problem, without the buffer constraint, could be solved using well-known techniques such as Lagrange Multipliers [101]. Additionally, examining the optimal solution for a sequence of length N with channel rate r, as obtained using the VA, shows that the allocation for large enough groups of n contiguous blocks ($n \ll N$) is very close to nr bits. See Fig. 2-8.

In this section, we will illustrate how these observations can be exploited in generating a nearly optimal but much faster solution for the buffer constrained problem by solving a series of allocation problems with budget nr and no buffer constraint. To this end, we now provide a brief review of the Lagrange Multiplier optimization technique.

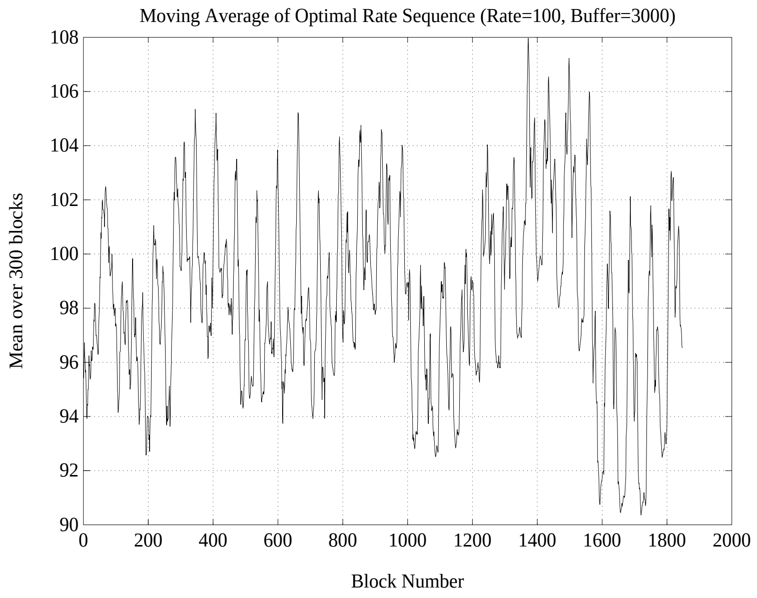


Figure 2-8: Moving Average of the Optimal Solution. The average is taken over 300 blocks. Note that the average is always close to the channel rate (100).

2.4.2.1 Constant slope optimization for budget–constrained allocation

Lagrangian optimization has been widely applied in the context of differentiable optimization in the presence of constraints [66, 67]. However, in [30], this method was also shown to be useful in the case of discrete constrained optimization. The concept was later used in [101, 117] to determine the optimal bit allocation, in a rate distortion sense, for a given source with an arbitrary discrete set of quantizers and, more recently, to find an optimal bit allocation for a wavelet packet decomposition [87]. A complete description of this technique goes beyond the scope of this chapter, so that we only summarize the main ideas, and refer the reader to [101, 30, 117, 87] for further details.

Consider the problem described in Formulation 2.2 without the buffer constraints (i.e. if the buffer size were unlimited) but instead with a rate budget constraint, so that R is the maximum total number of bits that can be used to code the sequence. We will first show how an efficient solution can be obtained for the budget–constrained problem. This solution will then be shown to be useful for our problem.

Formulation 2.3 *(Budget-Constrained optimization)*

The problem is to find the mapping x that solves:

$$\min(\sum_{i=1}^{N} d_{ix(i)}), \qquad \text{subject to :} \qquad \sum_{i=1}^{N} r_{ix(i)} \leq R, \tag{2.5}$$

where R is the total budget.

Now, the main result states that, for any real positive number λ, the Lagrange multiplier:

Theorem 2.1 *[30, 101] If the mapping $x^*(i)$ for $i = 1, 2, \ldots, N$, minimizes:*

$$\sum_{i=1}^{N} d_{ix(i)} + \lambda \cdot r_{ix(i)}, \tag{2.6}$$

then it is also the optimal solution to the problem of Formulation 2.3, for the particular case where the total budget is:

$$R = R(\lambda) = \sum_{i=1}^{N} r_{ix^*(i)}, \tag{2.7}$$

so that:

$$D(\lambda) = \sum_{i=1}^{N} d_{ix^*(i)} \leq \sum_{i=1}^{N} d_{ix(i)}, \tag{2.8}$$

for any x satisfying Equation (2.5) with R given by Equation (2.7).

Since we have removed the constraints, for a given operating "quality" λ, Equation (2.6) can be rewritten as:

$$\min(\sum_{i=1}^{N} d_{ix(i)} + \lambda r_{ix(i)}) = \sum_{i=1}^{N} \min(d_{ix(i)} + \lambda r_{ix(i)}), \tag{2.9}$$

Figure 2-9: For each block, minimizing $d_{ix(i)} + \lambda r_{ix(i)}$ for a given λ is equivalent to finding the point in the R-D characteristic that is "hit" first by a "plane wave" of slope λ.

From Theorem 2.1, for a fixed λ we obtain *the best possible solution that meets the budget constraint of Equation (2.7).* The budget for which we obtain a solution is itself a function of the chosen λ. Therefore, to find a solution to the initial problem, we need to iteratively change λ until we find the multiplier λ^* such that the total number of bits used $R(\lambda^*) = R$, within a convex hull approximation. It has been shown [101] that this is a fast convex search. If for some λ^* we have $R(\lambda^*) = R$, then the theorem guarantees that the solution to the unconstrained problem is also the solution to the constrained one. A fast way of searching for λ^* using the bisection algorithm, is now described. We again refer the reader to [30, 101, 117, 87] for a more detailed description.

Algorithm 2.2 *Lagrangian optimization:*

Step 0 *: Start with two values λ_u and λ_l such that $R(\lambda_u) \leq R \leq R(\lambda_l)$ where $R(\cdot)$ is as defined above.*

Step 1: *Set $\lambda_{next} = |\frac{D(\lambda_l)-D(\lambda_u)}{R(\lambda_l)-R(\lambda_u)}| + \epsilon$, where ϵ is an arbitrarily small positive number added to make sure that the smallest rate is picked if λ_{next} is a singular slope value.*

Step 2 *: Repeat the optimization of Equation (2.9) for $\lambda = \lambda_{next}$. If $R(\lambda_{next}) = R$, stop. Else if $R(\lambda_{next}) \geq R$ set $\lambda_l = \lambda_{next}$, or else if ($R(\lambda_{next}) \leq R$) set $\lambda_u = \lambda_{next}$. Go to* **Step 1**.

The main advantage of this algorithm is its efficiency. Indeed, our experiments verify that usually a relatively small number of iterations (typically less than 10) is sufficient for convergence in the problems we have considered. An analysis of the comparative complexities of the VA and a constant slope based algorithm for the buffer-constrained quantization problem is given in Section 2.4.3.

As pointed out earlier, a solution for budget R may not exist for any λ, but using the constant slope algorithm will at least yield a solution that is optimal for a budget $R(\lambda)$ that is slightly less than R, i.e. within a convex hull approximation of

R. Figure 2-10 shows all the constant slope solutions that do not produce overflow or underflow for the allocation of the first 128 blocks of a sequence. As all final buffer states are not attainable, it is clear that there are no constant slope solutions for all desired budgets. However, it must be emphasized that those "constant slope paths" that can be constructed using the above described optimization are *optimal* in the sense that there is no better way of coding the sequence if the initial and final stages had to be those determined by the constant slope path.

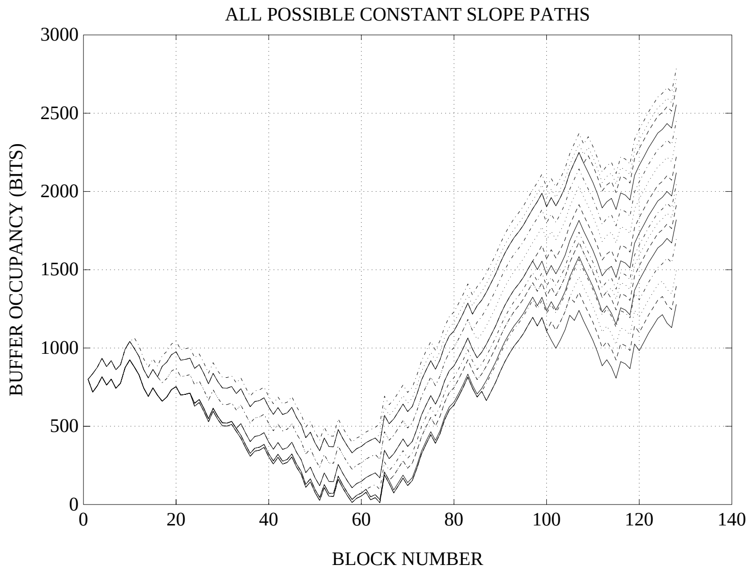


Figure 2-10: All possible "constant slope" solutions. Note that not all final states can be reached with a constant slope path. However, if a path exists, it is optimal. In this example $B_{max} = 3000$ bits.

2.4.2.2 Iterative constant slope optimization for buffer-constrained allocation

We now apply the above ideas to set up the following algorithm for the buffer-constrained allocation problem:

Algorithm 2.3 *Iterative constant slopes:*

Step 1 : *At every stage k, use Lagrangian optimization, with budget constraint $n \cdot r - B(k) + B_{max}/2$, to obtain the best non-buffer-constrained allocation for the following n blocks.*

Step 2 : *Using the quantizer indicated by the previous step for block k, release that block to the buffer and repeat* **Step 1** *for stage $k+1$.*

The budget constraint requires that the number of bits used for the next n blocks is such that at the n-th stage the buffer will be in mid-position ($B_{max}/2$). This is equivalent to performing a moving window optimization so that the bit rate of the k-th block depends only on the R-D characteristics of the following n blocks and on the buffer state at the k-th stage. Thus, we exploit both the finite memory of the problem (see results in Figure 2-7) and the effect of the buffer, in terms of getting the average bit rate of a finite number of blocks close to the channel bit rate.

Results show that this approach yields a solution very close to the optimal one, as obtained using the VA. When comparing the allocations obtained using Algorithms 2.1 and 2.3 for a typical sequence, experiments showed that a disparity in quantizer choice between the two approaches occurred for less than 5% of the signal blocks (see Fig. 2-11). Indeed, it can be seen that the optimal solution obtained with the VA follows a constant slope path for several blocks: the optimal solution is a *piecewise constant slope path*. Intuitively, as the optimal solution for a given budget without buffering is a constant slope solution, when buffering constraints are added, the solution becomes "piecewise constant slopes". Recent work [19] has studied using the lagrange multiplier as the control parameter of the encoder, and analyzed the stability of having a feedback function that sets λ given the buffer occupancy.

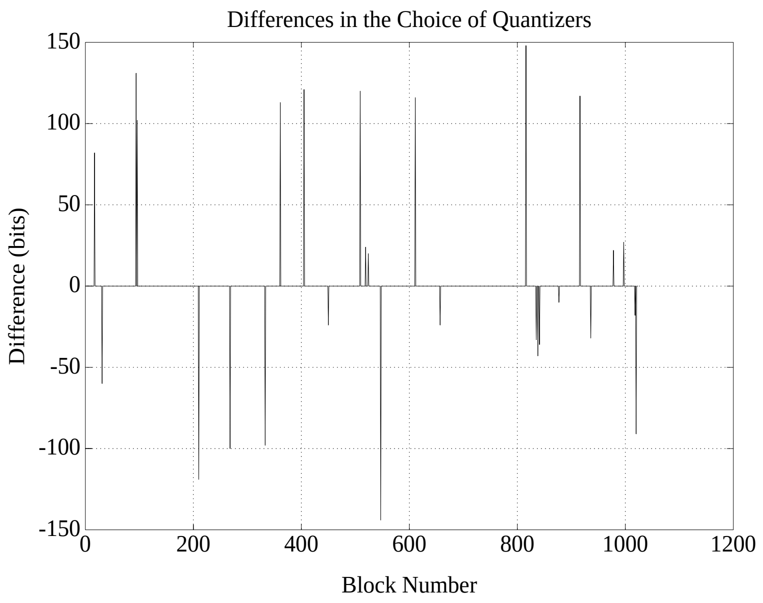


Figure 2-11: Differences in quantization choices between Algorithms 2.1 and 2.3. Note that the quantization obtained using iterative constant slopes and the VA is different for only a few blocks.

2.4.2.3 Fast iterative constant slope optimization based on heuristics

A straightforward way of improving the speed of the algorithm while sacrificing little quality is the following: use Algorithm 2.3, except perform optimization over n blocks only when the buffer size is above or below certain bounds. Call the heuristic threshold the fraction of the buffer size used as a bound (see Figure 2-12). Thus a "10% threshold" means that the path is recomputed whenever the buffer occupancy is below 10% or above 90% of the total buffer size. With this notation, a 50% threshold corresponds to using Algorithm 2.3. Note that while we use symmetric thresholds for simplicity, non-symmetric thresholds for overflow and underflow are also possible. The algorithm can be expressed in a more formal way as follows:

Algorithm 2.4 *Heuristic constant slopes (see Figure 2-12):*

Step 0 : *Initialize the stage count i to 1.*

Figure 2-12: An example of Algorithm 2.4. From block i to block j the allocation is not recomputed. Then, as the buffer state exceeds the threshold, the allocation is recomputed.

Results show that for typical sequences, for a heuristic threshold as low as 10%, for which the paths are seldom recomputed, more than 85% of the quantizer choices match those selected in the optimal solution. In Figure 2-13, the SNR of both the VA solution with limited memory and the heuristic (10%) approximation are compared. For a sufficiently large number of blocks, the heuristic approximation comes within 0.05 dB of the optimal value for our experiments.

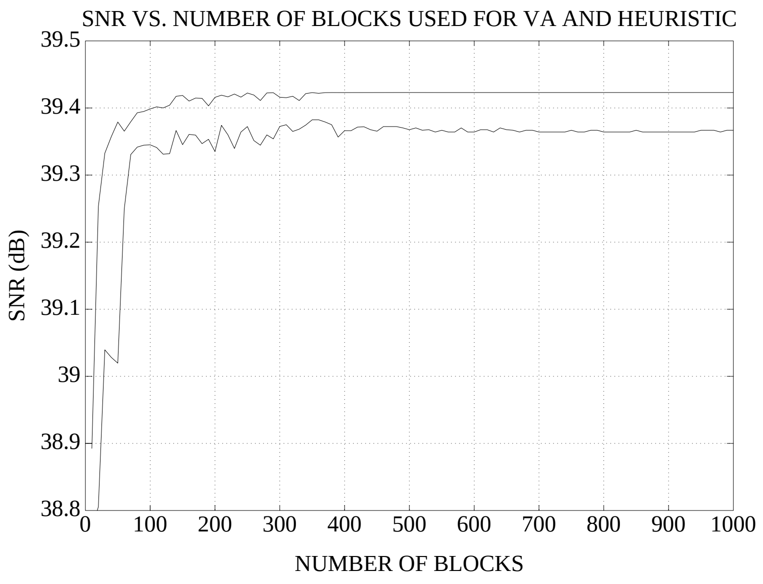


Figure 2-13: SNR of suboptimal VA (top) and Algorithm 2.4 with 10% heuristic threshold. When a sufficient number of blocks is considered, the heuristic approximation comes within 0.05 dB of the distortion for the optimal solution.

2.4.2.4 Generalized Lagrangian optimization

The Lagrangian optimization techniques techniques presented in the previous sections can be extended to multidimensional cases (i.e. those where there are more constraints that the budget constraint of (2.5)) using a vector Lagrange multiplier $\mathbf{\Lambda}$. See [32, 66, 67] for examples of problem solution using multiple lagrange multipliers. Consider the problem of Formulation 2.2 but now instead of writing a recursive computation of the buffer state, which includes a non-linear operator, substitute (2.4) with the set of $2 \cdot N$ equations:

$$0 \leq \sum_{i=1}^{k}(r_{ix(i)} - r) \leq B_{max} - r, \quad \forall k = 1 \cdots N, \tag{2.10}$$

where we assume for convenience that the set of quantizers allows us to always avoid underflow, so that the problem has a solution. Denote each of the constraints as:

$$g_{2k-1}(x) = \sum_{i=1}^{k}(-r_{ix(i)} + r) \leq 0 \tag{2.11}$$

and

$$g_{2k}(x) = -B_{max} + \sum_{i=1}^{k}(r_{ix(i)} - r) \leq 0, \tag{2.12}$$

which we will also represent as a $2N$ dimensional vector $\mathbf{g}(x)$. Then, associate a real positive coefficient $\mathbf{\Lambda}(i) = \lambda_i$ to each of the $2N$ constraints so that the Lagrangian Dual problem can be written as, for $\mathbf{\Lambda}$ a vector of size $2 \cdot N$ of positive real numbers.

Find $\mathbf{\Lambda}^*$ such that

$$w(\mathbf{\Lambda}^*) = \max_{\mathbf{\Lambda} \in R^{2N}, \lambda_k \geq 0, \forall k=1,\cdots,2\cdot N} w(\mathbf{\Lambda}) \tag{2.13}$$

where for each $\mathbf{\Lambda}$ we search for the solution x that minimizes

$$w(\mathbf{\Lambda}) = \min_x (\sum_{i=1}^{N} d_{ix(i)} + \sum_{k=1}^{N} \lambda_{2k-1} g_{2k-1}(x) + \lambda_{2k} g_{2k}(x)). \tag{2.14}$$

This approach generalizes that of Theorem 2.1 to the case where there is more than a single constraint. Thus we can see that the formulation of Theorem 2.1 is obtained when all λ_k are set to zero, except for λ_{2N}, i.e. when the only constraint is an upper bound on the total rate. $w(\mathbf{\Lambda})$ represents the solution that minimizes the cost for given $\mathbf{\Lambda}$ while $w(\mathbf{\Lambda}^*)$ is the optimal solution to our problem, once we have found the "correct" multiplier $\mathbf{\Lambda}^*$.

To change the value of $\mathbf{\Lambda}$ one can follow an iterative algorithm. Taking advantage of the fact that if x^j is the point at which the minimum of w is reached for $\mathbf{\Lambda}^j$, where the superscript j indicates the iteration number, then the vector $\mathbf{g}(x^j)$, where each component corresponds to one of the g_i functions, is a *subgradient* for $w(\lambda)$ at $\mathbf{\Lambda}^j$. The subgradient plays in this non-differentiable optimization a role analogous to that of the gradient in the optimization of differentiable functions. The subgradient indicates the direction in which the vector $\mathbf{\Lambda}$ has to change so that the maximum is attained. For instance in the next iteration

$$\mathbf{\Lambda}^{k+1} = \lambda^j + \rho^j \mathbf{g}(x^j) \tag{2.15}$$

where ρ^j is a stepsize which will determine the speed of convergence. See [67, 66] for a detailed study of convergence issues in this type of problems.

While this approach can potentially provide good solutions it may also be too complex in the case when the complete sequence is used. Also, if this approach is used there is no guarantee to achieve a global optimum since in the higher dimensional space we can not invoke the fast convex search that was used in the single multiplier formulation. However, this approach may be attractive when only a few frames are considered and has been recently proposed in [60], with results presented for the 2-frame case.

2.4.3 Complexity

In this section the following notation will be used: M is the number of quantizers available, N is the number of blocks to be coded, and B_{max} is the buffer size or, more precisely, the number of different states or nodes considered in the buffer (to allow for the case when clustering is performed).

2.4.3.1 Optimization using the VA

Since there are B_{max} nodes per stage, a total of $B_{max} \cdot N$ nodes have to be considered. Growing a branch to each individual node b involves choosing, among at most M

branches arriving to that node, the branch with minimum total cost, so that there are M comparisons per node. Denoting by $C_i(b)$ the cost accumulated up to stage i on the best path arriving at node b, we have that $C_i(b) = \min_{j=1,\cdots,M}(C_{i-1}(b - r_{ij} + r) + d_{ij}))$, where the minimum is over all M quantizers. The cost of each incoming branch is computed as the sum of the cost accumulated in the path up to the previous stage and the cost of the branch itself. Hence, for each node, the complexity increases linearly with the number of quantizers, M. Once the full trellis has been grown, determining the optimal path involves backtracking from the final stage to the initial stage. This requires one addition per stage, which we can neglect in estimating the order of complexity. The total complexity of growing the trellis is thus:

$$\mathcal{C}_v = \mathcal{O}(B_{max}NM) \tag{2.16}$$

Note that in the sliding window version of the VA (see Section 2.4.1.3) the complexity of generating the trellis remains the same but the cost of backtracking is no longer negligible, since backtracking has to be performed for every iteration.

2.4.3.2 Solution using constant slope optimization

Consider Algorithm 2.3, with n blocks being used for each iteration of the Lagrangian optimization. Finding the path for each set of n blocks requires, on the average, I iterations of Algorithm 2.2. In each iteration of Algorithm 2.2, i.e. for a fixed λ, one has to find, for each of the n blocks, the quantizer that minimizes $(r_{ij} + \lambda d_{ij})$, which as before makes the complexity increase linearly with M. Note that recomputing λ in Step 1 of each iteration of Algorithm 2.2 is negligible compared to the $n \cdot M$ comparisons required in Step 2. This procedure is repeated for each of the N blocks. Since I remains essentially independent of the other parameters of the problem, an estimate of the complexity is:

B_{max}		2000	3000	4000
$\log_2(B_{max})$		10.96	11.55	11.96
(i)	Time (s)	82.8	121.8	162.3
	SNR (dB)	37.93	37.96	37.98
(ii)	Time (s)	77.8	81.1	83.6
	SNR (dB)	37.92	37.93	37.95
(iii)	Time (s)	150.1	156.1	157.7
	SNR (dB)	37.92	37.94	37.96
(iv)	Time (s)	7.9	4.1	5.2
	SNR (dB)	37.91	37.90	37.84

Table 2.2: Comparison of distortion and execution time for: (i) VA, (ii) a constant slope optimization with length 200 and, (iii) length 400, and (iv) heuristic approximation with 10% threshold.

$$\mathcal{C}_s = \mathcal{O}(nNM) \tag{2.17}$$

In practice, values of n that yield a good performance, are of the order of $k\log(B_{max})$, where k is a constant, so that in general we have that: $n \ll B_{max}$.

2.4.3.3 Experimental comparison

To give a better idea of the trade-offs between complexity and performance involved in choosing one of the algorithms of Section 2.4, we now look at the results of a typical example (see Table 2.2). We fix the sequence, rate, and quantizer set and compute, for buffer sizes (in bits) of 2000, 3000 and 4000, the solution using the following algorithms: (i) Algorithm 2.1 (optimal solution), (ii) Algorithm 2.3 with $n = 200$ blocks ("short look-ahead"), (iii) Algorithm 2.3 with $n = 400$ blocks ("long look-ahead"), and (iv) Algorithm 2.4 with 10% heuristic.

With the obvious disclaimer that the precise values depend on the chosen sequence, some general observations can be made:

- The sub-optimal algorithms afford good approximations to the optimal values.
- As just derived, the computation time for the VA increases linearly with the the buffer size, while the increase is only logarithmic for the Lagrangian–based algorithms. In the limit, as the buffer size goes to infinity, we would be dealing with an unconstrained allocation problem, for which the VA is clearly non–practical, while Lagrangian approaches can be efficiently used [101, 117, 87].
- A comparison between the execution times of the VA and Algorithm 2.3 is unfair since Algorithm 2.3 suffers from the overhead required by a sliding window optimization. A fairer comparison would be between Algorithm 2.3 and the sliding window version of the VA. Our results indicate that the former is significantly faster than the latter.
- Clearly, the heuristic methods of Algorithm 2.4 have an edge in terms of speed, although the gains are very dependent on the sequence. In the limit, if a "constant slope solution" to the problem exists (i.e if the buffers play no role and the optimal solution is the same as in the unbuffered case for the same total rate) then Algorithm 2.4 would find the solution in one iteration!

2.5 Applications

This section is devoted to describing several specific applications of the theoretical analyses and algorithms presented so far. We first use an optimal solution to the buffer-constrained quantization problem to determine the appropriate buffer size for a given source and application. We also show the relevance of our methods in providing a benchmark for buffer control algorithms as well as to analyze the statistical gain in ATM networks. Our intention here is not to be exhaustive, but rather illustrative.

2.5.1 Optimal buffer size

The first question that arises in the study of our buffer constrained problem is that of the optimal buffer size. As we have seen, a budget–constrained unbuffered quantization provides the lower bound for the distortion incurred. It is of interest to examine what the minimal buffer size might be for which the buffer becomes "transparent" to the bit allocation problem. This is relevant in light of the different applications that were outlined in Section 2.2.3. While in some cases, like real–time services, the maximum buffer size that can be used in the system is limited by the total delay ΔT, it is apparent that in other applications, like video CD-ROM's, the memory to be used is essentially a design choice, with no limitation other than cost. In such a case, the buffer size need not be bigger than the maximum length of oscillation in buffer occupancy when a "constant slope" is maintained.

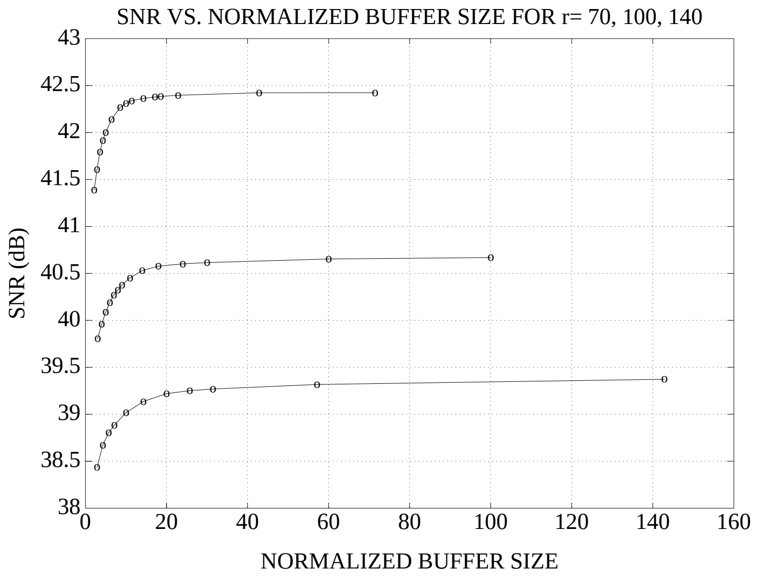


Figure 2-14: SNR vs. normalized buffer size. The normalized buffer size (B_{max}/r) represents the size of the memory expressed in number of blocks coded at the nominal rate of r bits. Note that for different values of r the curves have a similar shape and the optimal normalized buffer size is roughly the same.

In our experiments, for a given sequence, channel rate and set of quantizers, we find the optimal solution for several buffer sizes. As can be seen in Figure 2-14, the marginal gain of increasing the buffer size decreases as the buffer size gets large. This gain becomes zero once the maximum buffer size that a "constant slope" solution generates has been reached. The buffer size to choose, if the choice is possible, is that one for which the optimal value is "just" reached, and necessarily depends on the variability of the source. In the curves, in order to remove the dependence on the channel rate, the buffer size is normalized by the channel rate so that the horizontal axis represents the maximum number of blocks that could fit in the buffer if coded at the nominal rate. It can be seen that all three curves, corresponding to three channel rates for the same sequence, have the "knee" at roughly the same normalized buffer size. This result indicates that, as expected, the optimal buffer size is a function of the sequence to be coded. Note that the knee is reached first for the higher channel rates. This happens because the set of quantizers is maintained constant in all three experiments so that as rate increases there are less possible choices. In the limiting case, if the channel rate was above the maximum rate of the highest quality quantizer in the set, the optimal solution would be reached for any buffer size, since the buffer would be permanently in underflow.

Although one cannot guarantee that a given buffer size will be appropriate for a certain non-stationary video sequence, if the range of variability of the source is known, a good choice for the buffer size can still be made. Finally, it is worth noting that for some applications, the buffer size can indeed be tailored for each specific sequence. Consider, for instance, a workstation display of a previously coded sequence. In this environment, since delay is not a factor, it would be possible to decide on different buffer sizes depending on the specific characteristics of the sequence by prefixing, in the sequence header, the optimal buffer size that was chosen during the coding process.

2.5.2 Benchmarking

Many different applications require a real–time buffer control strategy, where complexity considerations may not permit the use of the methods of Section 2.4.2. In that case, given a sequence, we can compute the optimal buffer–constrained allocation, and thus have a *benchmark* serving as an upper bound for performance in assessing real–time algorithms. Typical examples would be workstation display, video on CD-ROM, buffered JPEG or MPEG. In this section, we present an example of the use of benchmarking for an activity–weighted buffered JPEG–like coding environment (unlike JPEG we allow different quantizer scales to be used for each block within a frame).

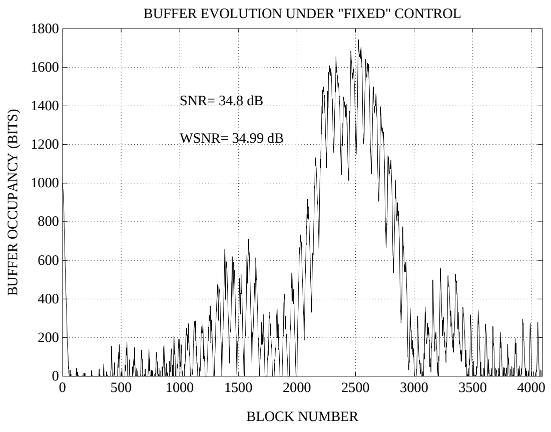
As discussed in Section 2.2.1, the only requirement we impose on the cost measure to be minimized is that it be additive. As an example, we consider a typical DCT–based coding scheme, such as JPEG, and take as an activity measure the sum of the absolute values of the DCT coefficients of each block. Then we choose as cost measure for each block the MSE divided by the activity (so that higher activity blocks are permitted to have a higher MSE).

We now consider a practical scheme that might be used, and assess its performance against the "optimal" benchmark provided by the VA. While distributing the allocated rate among the blocks in a frame, the criterion should tend to use finer quantizers for the lower activity blocks (for which coding errors are more noticeable) while coarser quantization can be afforded for the higher activity blocks. Given the nominal bit allocation budget per frame, the mean activity of the frame, $\bar{A}$, is computed and an activity classification is performed for each block i. Blocks with an activity index A_i close to the mean $\bar{A}$ are assigned the "nominal" quantizer, while those deviating from the mean are coded using finer quantizers (if their activity is less than average) or coarser quantizers (if their activity is more than average). Thus, for a real-time coding scheme, a fixed mapping is set up between the quantizer grade chosen and the relative activity ($A_i/\bar{A}$). Note that the nominal quantizer is chosen so that the number of bits actually allocated to the frame is close to the desired objective. This simple approach, however, does require an encoding delay of one frame, since $\bar{A}$, has to be determined.

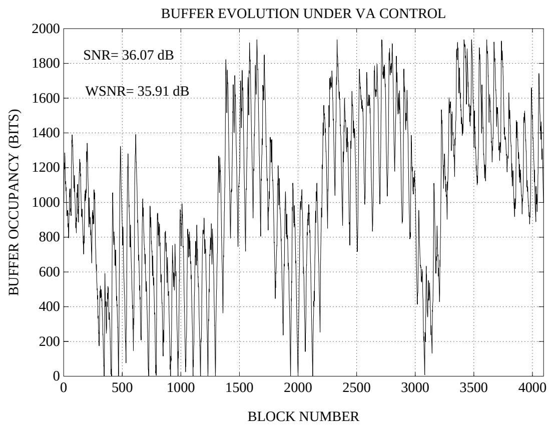
We can now estimate the suboptimality of such a rate control scheme when it is used in a finite buffer environment. Suppose the practical scheme described above is used with a buffer size B_{max} that just prevents overflow when the buffer empties out at a "channel rate" r (see Figure 2-15(a)). Note that for the example considered, in order to accommodate the coded stream in the buffer, we need to use a channel rate higher that the average coding rate of the source, due to the peakiness of the source. In Figure 2-15(b), we show how we can get an optimal upper bound to the performance given the same conditions (set of quantizers, r, B_{max}) by using the VA with an activity–weighted MSE cost measure. In our results, for the coding of one frame, VA proves to be better than a "fixed" mapping by 1 dB in both SNR and weighted SNR (WSNR).

2.5.3 Statistical multiplexing of ATM sources

Within an Asynchronous Transfer Mode (ATM) network, the user negotiates with the network the parameters of each connection, specifying the mean channel rate, maximum peak rate available to the user, etc.. [90]. See Chapter 3 for more details. Thus several VBR sources would be able to share transmission resources and buffers within the network, so that advantage can be taken of what has been called *statistical multiplexing gain* (SMG): the bit stream combining the multiplexed sources has a



(a)



(b)

Figure 2-15: Comparison of "fixed" activity–based allocation and optimal buffer–constrained allocation. In both figures the channel rate is $r = 64$ bits/block and the buffer size $B_{max} = 2000$ bits. $WSNR$ denotes the activity weighted SNR. (a) Buffer evolution for "fixed" control. For the chosen buffer size and nominal quantizer, there is underflow at both the beginning and the end of the frame. (b) Buffer evolution under VA allocation.

less peaky behavior (i.e. its "peak to mean rate" ratio is smaller than that of the individual sources) and therefore the probability of buffer overflow is lower. Studies of SMG for packet video [63, 109] have expressed the gain only in terms of reducing overflow probability, i.e. looking at the problem from a network point of view. In [37] it was shown how, for certain sources such as speech and video, end–to–end SNR is a more meaningful measure of performance than packet loss. This fact was used to propose a system of packet priorities so that under congestion the least "important" information, in the distortion sense, is lost first. We look at the problem of statistical multiplexing also from a source coding point of view (i.e. taking into account both rate and distortion), and show an example of how to use the previously described techniques to evaluate the maximum performance gain of source multiplexing in an ATM environment.

It has to be noted that our experiment does not take into account the fact that sources will be policed by the network [90] and thus no restrictions are imposed on the parameters (peak and mean rate) of each individual source being multiplexed. The problem of optimizing source coding under policing constraints is studied in Chapter 3. Our results provide a bound on the achievable performance given the set of sources and the total bit rate available to transmit them.

The experiment is necessarily source-dependent, and is given as an example of what a typical behavior might be. One could expect higher gains if the sources are highly uncorrelated (different sources having peaks during different time intervals) while gains would have to be modest for correlated sources. In fact, in the limit of high correlation, when all sources are identical, there would be no gain in using multiplexing.

In our experiment we consider 4 sources, each consisting of 1000 blocks, that are multiplexed into a buffer of size $4 \cdot B$ before being sent through a channel of rate $4 \cdot R$. As stated, this problem is equivalent to the one we have treated previously, so that we can use the VA to determine the optimal allocation given the buffer constraints. This can be done in two ways:

(i) Allocate the rate *independently* for each source, i.e. use the VA on each one assuming a rate of R and a buffer size of B. The VA minimizes the distortion of each source.

(ii) Perform the VA in a *combined* fashion, i.e., for a rate of $4 \cdot R$ and a buffer size of $4 \cdot B$. At each stage the algorithm determines the quantizers to be used in each of the sources, taking into account the combined rate distortion characteristic. With this approach, the VA minimizes the total distortion of the four sources.

Figure 2-16 shows the evolution of the multiplexing buffer under methods **(i)** and **(ii)** (dashed and solid lines respectively). Note that when we perform an independent allocation, *the common buffer is not fully used.* In contrast, when a combined allocation is performed, the buffer reaches near-full and near-empty levels during the

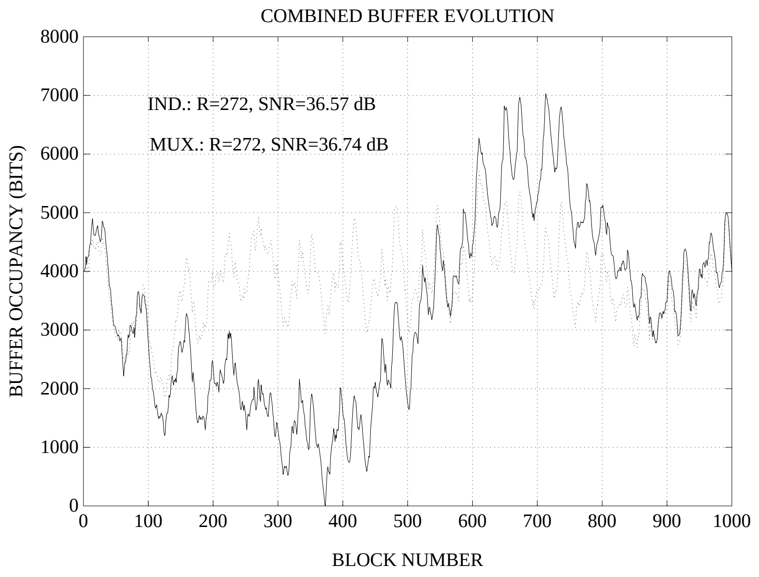


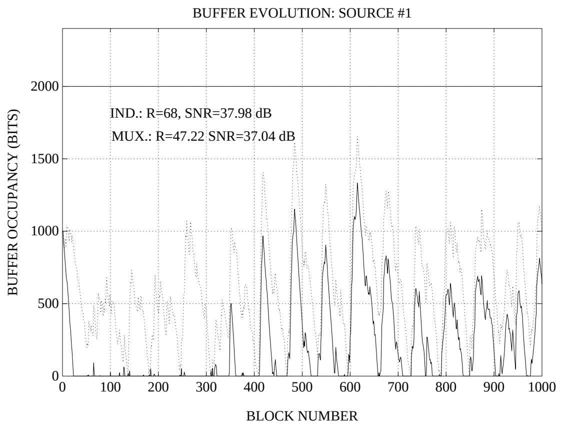
Figure 2-16: Buffer evolution in the common buffer for the 4 sources when the allocation is computed using a combined or multiplexed VA (solid line) or an independent VA (dashed line). The resulting total SNR (obtained by averaging the MSE of all sources) increases as was to be expected when a combined VA is used for the allocation. The independent VA uses 1/4 of the rate and buffer size of the combined scenario independently for each source.

simulation. For this particular example, the gain achievable by using a combined VA is nearly 0.2 dB for the average distortion. However, the individual distortions of each of the sources with either method are of more interest (see Figure 2-17). Our results show that in order to improve the overall quality, one source is allocated fewer bits by the combined algorithm (see Figure 2-17(a)), while the other three (Figure 2-17(b),(c),(d)) increase their total allocation. It can be seen that source 3 increases its SNR by almost 1 dB while source 1 loses close to 1 dB. Overall, minimizing the total distortion tends to average out the SNRs of the individual sources. Note that there is also a gain from the buffer size point of view: for sources 2, 3 and 4 to reach the same distortion values as with method (ii), the buffers used with method (i) would need to be much larger than the original size of $B_{max} = 2000$ bits, thus the same distortion can be achieved with less end-to-end delay (see Chapter 3).

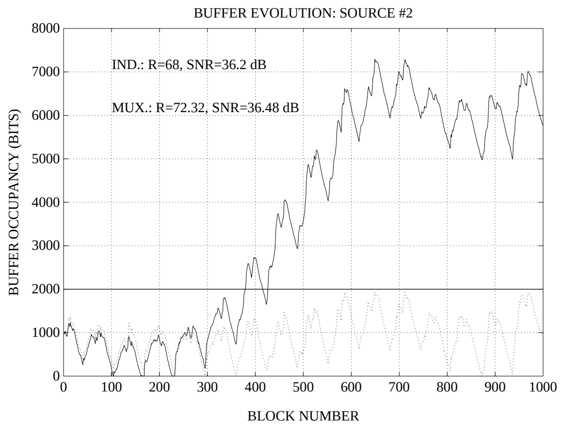
Finally, it should be pointed out that the ideas explained in this section are directly applicable to schemes where several video sources are multiplexed within a single CBR channel, such that each source can use a variable number of bits.

2.6 Conclusions

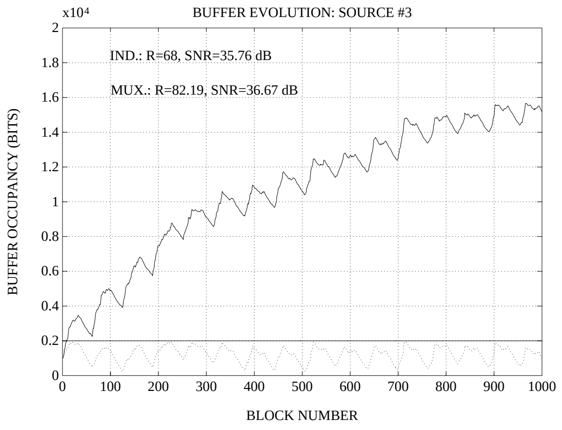
In this chapter, a global view of the problem of buffer-constrained quantization has been presented. A detailed study of the problem from a theoretical point of view is undertaken and, based on it, practical methods to compute the optimal solution to the problem are described. In particular, we present a class of nearly optimal fast algorithms exhibiting a substantial reduction in computational complexity. We have shown some possible applications for our algorithms, such as benchmarking for buffer control algorithms, determination of the optimal buffer size for a given coding environment, and analysis of statistical multiplexing gain in ATM networks. Additionally, our optimal algorithm is useful for applications where encoding delay



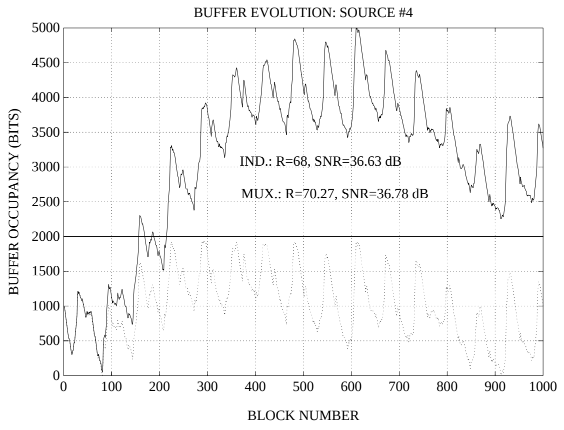
(a)



(b)



(c)



(d)

Figure 2-17: Individual buffer occupancies for each source. (a) Source 1, (b) Source 2, (c) Source 3, (d) Source 4. The independent buffer evolution (dashed line) corresponds to a buffer size of 2000. The combined or multiplexed buffer evolution (solid line) is simulated based on the combined allocation and assuming that the channel rate was the same as in the independent case, i.e. 1/4 of the total rate. Observe that the most demanding sources (2,3,4) increase their effective rates in the combined case at the expense of the least demanding source (1). Overall the same total rate is used and, as seen in Figure 2-16, the overall SNR is improved when doing a combined allocation.

is not important, like broadcast, video on CD–ROM, workstation display or buffer–constrained JPEG. The case of dependent quantization, that is when the bit rate of a given block depends both on the quantizer used and on the previous block (the simplest example would be interframe DPCM video sequence coding) will be examined in Chapter 4.