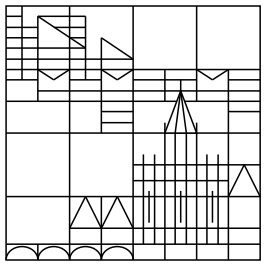

Universität Konstanz

On the complexity of rate-distortion optimal streaming of packetized media

Martin Röder
Jean Cardinal
Raouf Hamzaoui

Konstanzer Schriften in Mathematik und Informatik
Nr. 195, Januar 2004

ISSN 1430–3558

Universität Konstanz
Fach D 188, 78457 Konstanz, Germany
Email: preprints@informatik.uni–konstanz.de
WWW: http://www.informatik.uni–konstanz.de/Schriften/

On the complexity of rate-distortion optimal streaming of packetized media*

Martin Röder, Jean Cardinal, Raouf Hamzaoui

Martin Röder, Raouf Hamzaoui
Universität Konstanz
FB Informatik & Inf. Wiss.
Fach M 697
D-78457 Konstanz
Germany
{roeder,hamzaoui}@inf.uni-konstanz.de

Jean Cardinal
Université Libre de Bruxelles
Computer Science Dept.
CP 212
B-1050 Brussels
Belgium
jcardin@ulb.ac.be

Abstract

We consider the problem of optimal rate-distortion streaming of packetized multimedia data in the context of sender-driven transmission over a single-QoS network using feedback and retransmissions. For a single data unit, we prove that the problem is NP-hard and provide efficient branch and bound algorithms that are much faster than the previously best solution based on dynamic programming. For a group of data units, we show how to compute optimal solutions with branch and bound algorithms. The branch and bound algorithms for a group of data units are much slower than the current state of the art, a heuristic technique known as sensitivity adaptation. However, in many real-world situations, they provide a significantly better rate-distortion performance.

1 Introduction and preliminaries

One of the most popular frameworks for streaming packetized multimedia data over a packet erasure channel using feedback and retransmissions was introduced by Chou and Miao [1, 2]. This framework enables a precise mathematical formulation of the problem of how and when to transmit a group of interdependent data units before a delivery deadline in a rate-distortion optimal way. It shows that rate-distortion optimal transmission policies can be obtained by minimizing a Lagrangian. Chou and Miao [1, 2] first provide a dynamic programming algorithm to compute an optimal policy when a single data unit has to be transmitted. For the transmission of a group of interdependent data units, they propose a heuristic iterative descent method called sensitivity adaptation (SA), which uses the dynamic programming algorithm to optimize for one data unit at a time, keeping the other data units fixed.

Chou and Miao's framework was also successfully used to handle other transmission scenarios [3, 4, 5, 6]. In this paper, we focus on sender-driven transmission over a single-QoS network [1, 2]. We first show the NP-hardness of the problem for a single data unit (Section 2.1). In Section 2.2, we provide a branch and bound algorithm to minimize the Lagrangian for a single data unit. Our algorithm is in practice much faster than the dynamic programming approach

*This work will be presented at DCC'04, the Data Compression Conference, Snowbird, Utah, March 2004.

of Chou and Miao [1, 2]. Thus, by using our branch and bound algorithm instead of dynamic programming, one can significantly speed up the SA algorithm. Moreover, in contrast to Chou and Miao who find only solutions whose error-cost points are on the lower convex hull of the admissible points, we provide another branch and bound algorithm that can compute optimal solutions that are unreachable with the Lagrange method. In Section 3, we show that for a policy of a group of data units to be optimal, the policy of each data unit must also be optimal. One immediate consequence is that the problem of optimal streaming of a group of data units is also NP-hard. Then we give branch and bound algorithms for computing optimal policies for a group of data units. Section 4 gives experimental results for a single and a group of data units.

In the remaining of this section, we recall the terminology and notation needed to describe the sender-driven transmission over a single-QoS network context [1, 2]. A multimedia source (e.g., audio, image, video) is encoded and packetized into a finite set of *data units*. The inter-dependency between the data units is given by a directed acyclic graph (V, E), where the set of vertices V is the set of data units and the set of edges $E \subset V \times V$ is given by $(l', l) \in E$ if l' must be decoded so that l can also be decoded. For $(l', l) \in E$, we write $l' \prec l$ and say that l' is an *ancestor* of l. We also use the notation $l' \preceq l$ if $l' \prec l$ or $l' = l$. We denote by D_0 the distortion if no data unit is decoded, $B_l > 0$ the size in bytes of data unit l, and ΔD_l the amount by which the distortion is decreased if data unit l is decoded compared to the distortion if only the ancestors of l are decoded. A channel packet containing a data unit and sent at time s can be either lost with probability ϵ_F, independent of s, or received at time s', where the *forward trip time* $FTT = s' - s$ is a random variable with probability density function p_F. Similarly, when a packet is sent from the client to the server through the feedback channel, it is either lost with probability ϵ_B or received after a *backward trip time* BTT, which is a random variable with probability density function p_B. We also define the *round trip time* RTT as the sum $FTT + BTT$. For convenience, by setting $FTT = \infty$ when a data unit is lost, one can extend the random variable FTT to make it include packet loss [1]. In this situation, the cumulative distribution function of the extended random variable, $\overline{FTT}$ is

$$P\{\overline{FTT} \leq \tau\} = \int_0^\tau (1 - \epsilon_F) p_F(t) dt.$$

The same extension can be done for BTT, yielding an extended random variable, $\overline{BTT}$. Finally, $\overline{RTT}$ will denote the sum $\overline{FTT} + \overline{BTT}$.

Given a number of transmission opportunities at times $s_0, s_1, \ldots, s_{N-1}$ and a delivery deadline s_{DTS}, a transmission policy $\pi = (\pi(0), \pi(1), \ldots, \pi(N-1)) \in \{0,1\}^N$ of length $\text{len}(\pi) = N$ is used to describe the transmission of a data unit. Here $\pi(i) = 0$ means that the data unit should not be sent at opportunity $i \in \{0, \ldots, N-1\}$, whereas $\pi(i) = 1$ means that the data unit should be sent at opportunity i if no acknowledgment packet was received from the feedback channel before time s_i, and that the data unit should not be sent at this opportunity otherwise.

We define an error $\epsilon(\pi)$ for policy π as the probability that the data unit does not reach its destination before time s_{DTS}, that is,

$$\epsilon(\pi) = \prod_{i:\pi(i)=1} P\{\overline{FTT} > s_{DTS} - s_i\}. \tag{1}$$

We also define a cost $\rho(\pi)$ for policy π as the expected number of data unit transmissions. One can show (see Appendix 1) that

$$\rho(\pi) = \sum_{i:\pi(i)=1} \left(\prod_{j<i:\pi(j)=1} P\{\overline{RTT} > s_i - s_j\} \right). \tag{2}$$

The first problem discussed in this paper is the computation of rate-distortion optimal policies for the transmission of a data unit. We say that a policy π^* is optimal if there exists no policy π such that $\epsilon(\pi) \leq \epsilon(\pi^*)$ and $\rho(\pi) < \rho(\pi^*)$. Note that a policy π^* is optimal if and only if there exists $\epsilon > 0$ such that π^* minimizes $\rho(\pi)$ subject to the constraint $\epsilon(\pi) \leq \epsilon$. Chou and Miao [2] claim that computing the set of optimal policies is infeasible and propose to determine instead only optimal policies π for which the cost-error points $(\rho(\pi), \epsilon(\pi))$ are on the lower convex hull [7] of the set $\{(\rho(\pi), \epsilon(\pi)),\ \pi \in \{0,1\}^N\}$. We call these policies *convex hull* policies. Convex hull policies can be found by minimizing the Lagrangian

$$J_\lambda(\pi) = \epsilon(\pi) + \lambda\rho(\pi) \tag{3}$$

for all $\lambda > 0$. For a given λ, brute force minimization of the Lagrangian requires $O(N^2 2^N)$ time. Chou and Miao [2] reduce the complexity of the minimization problem to $O(N2^N)$ with dynamic programming.

We also consider the problem of rate-distortion optimal transmission of a group of interdependent data units. The transmission policies for the group of data units is described by a policy vector $\vec{\pi} = (\pi_1, \ldots, \pi_L)$, where π_i, $i \in \{1, \ldots, L\}$ is the transmission policy for the ith data unit of the group. The expected transmission cost (the expected rate) for $\vec{\pi}$ is

$$R(\vec{\pi}) = \sum_{l=1}^{L} B_l \rho(\pi_l). \tag{4}$$

By assuming independence of the transmission processes, the expected distortion for $\vec{\pi}$ is

$$D(\vec{\pi}) = D_0 - \sum_{l=1}^{L} \Delta D_l \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})). \tag{5}$$

A policy vector $\vec{\pi}^*$ is optimal if there exists no policy vector $\vec{\pi}$ such that $D(\vec{\pi}) \leq D(\vec{\pi}^*)$ and $R(\vec{\pi}) < R(\vec{\pi}^*)$. Here also Chou and Miao [2] propose to compute only optimal policy vectors $\vec{\pi}$ for which the rate-distortion points $(R(\vec{\pi}), D(\vec{\pi}))$ are on the lower convex hull of the set $\{(R(\vec{\pi}), D(\vec{\pi})),\ \vec{\pi} \in (\{0,1\}^N)^L\}$. We call these policy vectors convex hull policy vectors. Convex hull policy vectors can be found by minimizing the Lagrangian

$$J_\lambda(\vec{\pi}) = D(\vec{\pi}) + \lambda R(\vec{\pi}) = D_0 + \sum_{l=1}^{L} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \tag{6}$$

for all $\lambda > 0$. To minimize J_λ for a given λ, Chou and Miao [2] apply the SA algorithm, a heuristic iterative descent method. The idea is to minimize $J_\lambda(\pi_1, \ldots, \pi_L)$ one policy at a time, keeping the other policies fixed. Starting from an initial policy vector $\vec{\pi}^{(0)} = (\pi_1^{(0)}, \ldots, \pi_L^{(0)})$, where $\pi_1^{(0)} = \cdots = \pi_L^{(0)} = (1, \ldots, 1)$, a sequence of policy vectors $\vec{\pi}^{(k)} = (\pi_1^{(k)}, \ldots, \pi_L^{(k)})$, $k = 1, 2, \ldots$, is computed as follows. Select $l_k \in \{1, \ldots, L\}$. For $i \neq l_k$, set $\pi_i^{(k)} = \pi_i^{(k-1)}$ and for $i = l_k$, let

$$\pi_i^{(k)} = \arg\min_{\pi_l} J_\lambda(\pi_1^{(k)}, \ldots, \pi_{l-1}^{(k)}, \pi_l, \pi_{l+1}^{(k)}, \ldots, \pi_L^{(k)}) \tag{7}$$

$$= \arg\min_{\pi_l} S_l^{(k)} \epsilon(\pi_l) + \lambda B_l \rho(\pi_l), \tag{8}$$

where

$$S_l^{(k)} = \sum_{l \preceq l'} \Delta D_{l'} \prod_{l'' \prec l'} (1 - \epsilon(\pi_{l''}^{(k)})).$$

Since the sequence $J_\lambda(\vec{\pi}^{(k)})$ is nonincreasing and bounded below by zero, it converges to a local minimum. The complexity of the SA algorithm is $O(n_i N 2^N)$, where n_i is the number of iterations of the algorithm.

2 Single data unit

2.1 NP-hardness

We show that finding the best transmission policy for a single data unit is NP-hard by using a reduction from the knapsack problem [8]. We first formally define the problem POLICY.

Definition 1 (POLICY). *An instance of problem* POLICY *consists of a positive integer N, $N+1$ real numbers $s_0 \leq s_1 \leq \ldots \leq s_{N-1} \leq s_{DTS}$, two real random variables FTT and RTT with $FTT \leq RTT$, and a number $\epsilon^* \in [0,1]$. An optimal solution of the problem is a policy $\pi = (\pi(0), \pi(1), \ldots, \pi(N-1)) \in \{0,1\}^N$ satisfying $\epsilon(\pi) \leq \epsilon^*$ and minimizing the cost $\rho(\pi)$.*

For simplicity, and without loss of generality, we assumed in Definition 1 that the optimality constraint is slightly less restrictive than in Section 1. We also assumed that the loss probabilities ϵ_F and ϵ_B are equal to zero. Note that although we included the random variables in the definition, the problem is essentially combinatorial: the finite set of probabilities $P\{FTT > s_{DTS} - s_i\}$ and $P\{RTT > s_i - s_j\}$ is the only information we actually need about FTT and RTT.

Lemma 1. *Let π^* be an optimal solution for a given instance of problem* POLICY *with $\pi^* \neq (0, \ldots, 0)$. Then there exists an optimal solution π for the same instance with $\pi(0) = 1$.*

Proof. Let π^* be an optimal policy such that $\pi^*(0) = 0$. Let π be a policy identical to π^*, except that $\pi(0) = 1$, and the first element of π^* equal to one is set to zero in π. Let k be the index of this element.

We have $\rho(\pi) \leq \rho(\pi^*)$, since the probability $P\{RTT > s_i - s_k\}$ is replaced by the probability $P\{RTT > s_i - s_0\}$, which is never greater. We also have $\epsilon(\pi) \leq \epsilon(\pi^*)$, because $P\{FTT > s_{DTS} - s_0\} \leq P\{FTT > s_{DTS} - s_k\}$.

Hence π is not worse than π^*, and is optimal too. □

Definition 2 (KNAPSACK). *An instance of problem* KNAPSACK *consists of a positive integer m, positive real values w_i and v_i for each $i \in \{1, 2, \ldots, m\}$, and a positive real number B. An optimal solution of* KNAPSACK *is a set $S \subseteq \{1, 2, \ldots, m\}$ satisfying*

$$\sum_{i \in S} w_i \geq B, \tag{9}$$

and minimizing

$$\sum_{i \in S} v_i. \tag{10}$$

Theorem 1. POLICY *is NP-hard .*

Proof. We prove the result by constructing a polynomial transformation from KNAPSACK into POLICY. Without loss of generality, we consider a restricted version of KNAPSACK with the following additional assumptions: the values v_i and w_i are decreasing with respect to i, and $v_i \in [0,1]$ for all i. It is not difficult to show that KNAPSACK remains NP-hard under these restrictions [8, 9].

Let $(m, \{v_i\}, \{w_i\}, B)$ be such an instance of KNAPSACK. The corresponding instance of POLICY is defined by

$$\begin{aligned} N &= m+1, \\ s_0 &= -\delta, \\ s_i &= \frac{i}{N},\ i \in \{1, 2, \ldots, N-1\} \\ s_{DTS} &= 1, \end{aligned}$$

for some value $\delta > 1$, and by two random variables FTT and RTT whose cumulative distribution functions $P\{FTT \leq x\}$ and $P\{RTT \leq x\}$ satisfy

$$P\{FTT \leq 1 - \frac{i}{N}\} = 1 - 2^{-w_i},\ i \in \{1, 2, \ldots, N-1\}, \quad (11)$$

$$P\{FTT \leq \delta + 1\} = 1 - \mu, \quad (12)$$

$$P\{RTT \leq \delta\} = 0, \quad (13)$$

$$P\{RTT \leq \delta + \frac{i}{N}\} = 1 - v_i,\ i \in \{1, 2, \ldots, N-1\} \quad (14)$$

for some value μ such that $0 < \mu < \min\{2^{-w_1}, v_{N-1}, 2^B\}$. The values $P\{FTT \leq x\}$ and $P\{RTT \leq x\}$ that are not defined above can be chosen by any kind of continuous nondecreasing interpolation between the defined values. We also let their limits as x tends to infinity and to zero be equal to one and zero, respectively. This is illustrated in Fig. 1.

It is not difficult to check that the two cumulative distribution functions defined in this way are well-behaved nondecreasing functions taking values in $[0, 1]$. In particular, since the w_i are sorted in decreasing order, the values of $P\{FTT \leq x\}$ from (11) are increasing with x. Since μ is chosen so that $\mu \leq 2^{-w_1}$ this is true with respect to (12), too. The cumulative distribution function $P\{RTT \leq x\}$ satisfying (13)-(14) is also increasing if the v_i are sorted in decreasing order.

Next, we must check that given these two cumulative distribution functions, there exist two random variables FTT and RTT such that $FTT \leq RTT$. From the stochastic order relation ([10], Theorem A, p. 6), a sufficient condition is

$$P\{FTT \leq x\} \geq P\{RTT \leq x\}.$$

This can be ensured by setting:

$$\begin{aligned} P\{FTT \leq \delta + 1\} &\geq P\{RTT \leq \delta + \frac{i}{N}\},\ i \in \{1, 2, \ldots, N-1\}, \\ P\{FTT \leq \delta + 1\} &\geq P\{RTT \leq \delta + \frac{N-1}{N}\}, \\ 1 - \mu &\geq 1 - v_{N-1}, \\ \mu &\leq v_{N-1}, \end{aligned}$$

which is guaranteed by the choice of μ.

Finally, we set

$$\epsilon^* = \frac{\mu}{2^B},$$

which belongs to $[0, 1]$ from the choice of μ again.

Now we show that this instance of POLICY is equivalent to the original instance of KNAPSACK. Hence an optimal solution π of the first problem will yield an optimal solution of the second

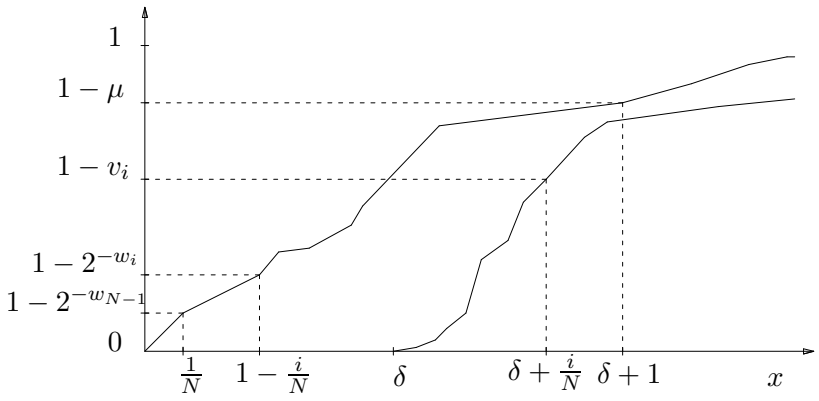

Figure 1: Illustration of the distributions used in the proof. The top curve is $P\{FTT \leq x\}$, and the bottom curve is $P\{RTT \leq x\}$.

simply by letting $S = \{i > 0 : \pi(i) = 1\}$, and conversely, the chosen subset S in the solution of KNAPSACK yields an optimal solution π for POLICY.

Let us first detail (1) with respect to the assignments above:

$$\begin{aligned}
\epsilon(\pi) = \prod_{i:\pi(i)=1} P\{FTT > s_{DTS} - s_i\} &\leq \epsilon^*, \\
(1 - P\{FTT \leq \delta + 1\}) \prod_{i>0:\pi(i)=1} \left(1 - P\{FTT \leq 1 - \frac{i}{N}\}\right) &\leq \frac{\mu}{2^B}, \\
\mu \prod_{i>0:\pi(i)=1} 2^{-w_i} &\leq \frac{\mu}{2^B}, \\
\prod_{i>0:\pi(i)=1} 2^{-w_i} &\leq \frac{1}{2^B}, \\
\sum_{i>0:\pi(i)=1} w_i &\geq B. \qquad (15)
\end{aligned}$$

We used Lemma 1 in the second line to include the probability $P\{FTT > s_{DTS} - s_0\}$ in the product. The last line is obtained by taking the negative logarithm on each side.

From (13)-(14) we have

$$P\{RTT > s_i - s_j\} = \begin{cases} v_i \text{ if } j = 0 \\ 1 \text{ otherwise.} \end{cases}$$

Now if we expand (2) with respect to the previous assignments and apply Lemma 1 again, we obtain

$$\begin{aligned}
\rho(\pi) &= \sum_{i:\pi(i)=1} \left(\prod_{j<i:\pi(j)=1} P\{RTT > s_i - s_j\} \right) \\
&= 1 + \sum_{i>0:\pi(i)=1} v_i. \qquad (16)
\end{aligned}$$

Inequality (15) and equation (16) are the same as (9) and (10), except for the constant additional term in (16), which does not modify the optimal solution. Hence POLICY is at least as hard as KNAPSACK. □

2.2 Branch and bound

In this section, we propose a class of branch and bound algorithms to compute convex hull and optimal policies. The algorithms exploit the following important bounds.

Suppose that a policy π of length N begins with prefix π'. That is, $\pi'(i) = \pi(i)$ for $i = 0, \ldots, \text{len}(\pi') - 1$ and $\text{len}(\pi') \leq N$. Since $\epsilon(\pi)$ is smallest if we transmit as often as possible, a lower bound for $\epsilon(\pi)$ is

$$\epsilon_{\min}(\pi') = \epsilon((\pi'(0), \pi'(1), \ldots, \pi'(\text{len}(\pi') - 1), \underbrace{1, \ldots, 1}_{N - \text{len}(\pi') \text{ times}})).$$

On the other hand, $\rho(\pi)$ is smallest if we do not transmit at all, thus a lower bound for $\rho(\pi)$ is

$$\rho_{\min}(\pi') = \rho((\pi'(0), \pi'(1), \ldots, \pi'(\text{len}(\pi') - 1), \underbrace{0, \ldots, 0}_{N - \text{len}(\pi') \text{ times}})).$$

A combination of $\epsilon_{\min}(\pi')$ and $\rho_{\min}(\pi')$ gives a lower bound for the expected Lagrangian $J_\lambda(\pi)$ of a policy π with prefix π':

$$J_{\lambda,\min}(\pi') = \epsilon_{\min}(\pi') + \lambda \rho_{\min}(\pi').$$

2.2.1 Convex hull policies

Using the above bound, we first propose a branch and bound algorithm to compute a convex hull policy for a fixed Lagrange multiplier λ. The policy prefixes of size n are recursively extended to policy prefixes of size $n + 1$. The recursive calls are stopped as soon as the lower bound $J_{\lambda,\min}$ exceeds $J_\lambda(\pi)$, where π is the currently best transmission policy π, or the size of a policy prefixes reaches N. In the latter case, π is updated with the policy prefix of size N.

More precisely, we initialize a policy π with an approximation of the optimal policy. This could be the solution of a previous optimization with similar parameters, or an arbitrary policy, e.g. $(0, \ldots, 0)$. The recursive part of the algorithm starts with an empty policy prefix π'. A single recursion on a policy prefix π' is as follows. We determine all possible policy prefixes π'_k whose length is the length of π' increased by 1. For the single policy Lagrange optimization, there are only two such policy prefixes (one can append either 0 or 1 to π'). Let π'_0 be π' with 0 appended, and π'_1 be π' with 1 appended. We then calculate $J_{\lambda,\min}(\pi'_k)$ for $k = 0, 1$. If $J_{\lambda,\min}(\pi'_k) > J_\lambda(\pi)$, we know that no policy with prefix π'_k can be better than π. Otherwise, if $\text{len}(\pi'_k) = N$, we found a better policy than π; so we set $\pi = \pi'_k$. If $\text{len}(\pi'_k) < N$, the described recursion is done again on π'_k. If the recursion has to be done on both π'_0 and π'_1, it should be done in increasing $J_{\lambda,\min}(\pi'_k)$ order. The optimal policy will be in π when the algorithm ends. A pseudo-code of the algorithm is given in Appendix 2.

It is straightforward to update the error bound $\epsilon_{\min}$ at each recursive step in constant time, but for the cost bound $\rho_{\min}$, the time complexity of an update is linear. The worst-case complexity of the algorithm is therefore $O(N2^N)$, since the number of recursive calls is bounded by $O(2^N)$. However, in practice, the pruning allows a substantial speed-up.

We now give a branch and bound algorithm to compute all convex hull policies in a single run. The main task will be to test if the cost-error point of a policy with a given prefix belongs to Γ, the lower convex hull of the points already checked, and to update Γ with insertions and deletions. For simplicity, we will identify a policy with the cost-error point associated to it.

We assume that Γ is sorted in nondecreasing cost and nonincreasing error order. This can be easily done, e.g., with a binary search tree. We denote the ith policy in Γ by $\Gamma(i)$; so we assume

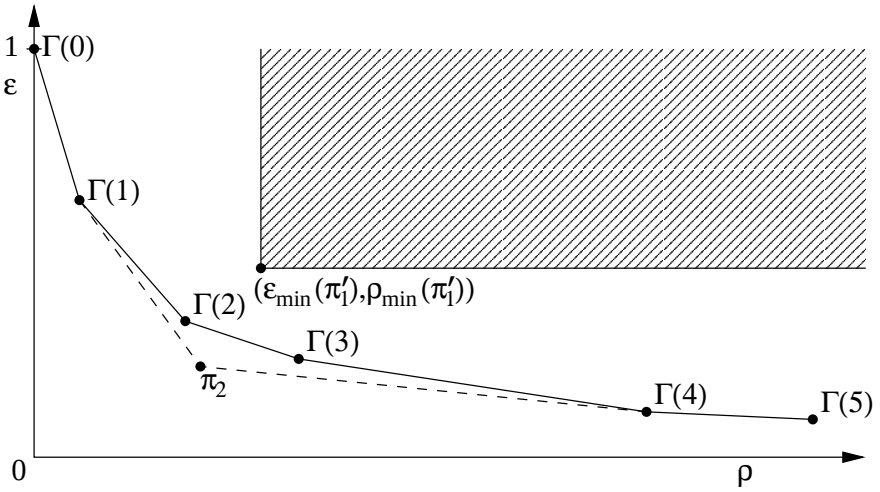

Figure 2: Example of convex hull test and insertion

that the ordering satisfies $\rho(\Gamma(i)) \leq \rho(\Gamma(i+1))$, $i = 0, \ldots, |\Gamma| - 2$ and $\epsilon(\Gamma(i)) \geq \epsilon(\Gamma(i+1))$, $i = 0, \ldots, |\Gamma| - 2$. Then, we define a function *CHTest(*Γ*,*π'*)* that tests if a policy with prefix π' belongs to the lower convex hull Γ. The test consists of the following steps:

1. Try to find k such that $\rho(\Gamma(k)) < \rho_{\min}(\pi') \leq \rho(\Gamma(k+1))$.
2. If no such k exists, set *CHTest(*Γ*,*π'*)*="yes".
3. If $(\rho_{\min}(\pi'), \epsilon_{\min}(\pi'))$ is above the line joining the points $\Gamma(k)$ and $\Gamma(k+1)$, set *CHTest(*Γ*,*π'*)*="yes", otherwise set it to "no".

To insert a new policy into Γ, we define a procedure *CHInsert(*Γ*,*π*)*. We assume that *CHTest(*Γ*,*π*)*="yes" at the time we call *CHInsert(*Γ*,*π*)*. The procedure first inserts π into Γ and then convexifies Γ by ensuring that the lines joining neighboring points have an increasing slope. This can be done by checking if the left neighbor of the inserted point is above the line joining the left neighbor of the left neighbor of the inserted point to the inserted point. If it is, the left neighbor of the inserted point is removed and the check is repeated. The same check is done on the right neighbors of the inserted point.

Figure 2 illustrates *CHTest* and *CHInsert* on an example. Suppose that Γ currently consists of $\Gamma(0), \ldots, \Gamma(5)$ and that we want to test if the policies with prefix π'_1 belong to the lower convex hull. *CHTest(*Γ*,*π'_1*)* equals "no", because π'_1 is above the line joining $\Gamma(2)$ and $\Gamma(3)$. Since all policies with prefix π'_1 are in the shaded area, none of them is in the lower convex hull. Suppose we want to insert π_2 into Γ. *CHInsert(*Γ*,*π_2*)* removes $\Gamma(2)$ and $\Gamma(3)$ from Γ, because they are above the lines connecting $\Gamma(1)$ and π_2 and π_2 and $\Gamma(4)$, respectively.

The branch and bound method to compute all convex hull policies is similar to the branch and bound method for the Lagrange optimization. We initialize Γ (the current lower convex hull) with $\Gamma = \{(0, \ldots, 0), (1, \ldots, 1)\}$. We then start the recursive part of the algorithm with prefix $\pi' = (1)$. In each single recursion, we determine π'_k, $k = 0, 1$ as in the Lagrange optimization. For each π'_k, we then determine *CHTest(*Γ*,*π'_k*)*, and if it equals "yes", we either insert π'_k into Γ with *CHInsert(*Γ*,*π'_k*)* if $\text{len}(\pi'_k) = N$, or we do the recursion again on π'_k if $\text{len}(\pi'_k) < N$.

When the algorithm stops, Γ is the lower convex hull of the policies with length N. A pseudo-code of the algorithm is given in Appendix 3.

2.2.2 Optimal policies

The branch and bound approach to compute the convex hull policy for a fixed Lagrange multiplier can also be adapted to find an optimal policy for a given maximum cost $\rho_{\max}$. In each recursion, we discard a policy prefix π' if $\epsilon_{\min}(\pi') > \epsilon(\pi)$ or $\rho_{\min}(\pi') > \rho_{\max}$ or $\text{len}(\pi') = N$, $\epsilon_{\min}(\pi') = \epsilon(\pi)$, and $\rho_{\min}(\pi') > \rho(\pi)$. Here π denotes the current best policy.

To find all optimal points, we use the same algorithm as for the lower convex hull. We only need to change test function *CHTest* and insertion procedure *CHInsert*. We now use a test function *OPTest(*Π,π'*)* that checks if policies with prefix π' are optimal among the set $\Pi = \{\Pi(0), \ldots, \Pi(|\Pi| - 1)\}$ of all current optimal policies. This is done as follows.

1. Try to find $k \in \{0, \ldots, |\Pi| - 1\}$ such that $\rho(\Pi(k)) < \rho_{\min}(\pi') \leq \rho(\Pi(k+1))$.

2. If no such k exists, $\rho_{\min}(\pi')$ is either smaller or larger than all $\rho(\Pi(k))$. If it is smaller, set *OPTest(*Π,π'*)*="yes". If it is larger, set it to "yes" if $\epsilon_{\min}(\pi') < \epsilon(\Pi(|\Pi| - 1))$ and to "no" otherwise.

3. If such a k exists, set *OPTest(*Π,π'*)*="yes" if $\epsilon_{\min}(\pi') < \epsilon(\Pi(k))$ and to "no", otherwise.

To insert a new policy into the current set of optimal policies Π, we define a procedure *OPInsert(*Π,π*)* on Π and a policy π. Here we also assume that *OPTest(*Π,π*)*= "yes" when calling *OPInsert(*Π,π*)*.

The procedure first finds the smallest $k \in \{0, 1, \ldots, |\Pi| - 1\}$ such that $\rho(\pi) \leq \rho(\Pi(k))$. If no such k exists, it sets $k = |\Pi|$. Then, π is inserted at the end of Π if $k = |\Pi|$, at the beginning of Π if $k = 0$, and between $\Pi(k-1)$ and $\Pi(k)$, otherwise. Note that after the insertion, we have $\pi = \Pi(k)$. Finally, all $\Pi(i)$ with $i > k$ and $\epsilon(\Pi(i)) \geq \epsilon(\pi)$ are removed from Π. Then all $\Pi(i)$ with $i > k$ and $\epsilon(\Pi(i)) \geq \epsilon(\Pi(k))$ are removed from Π. A pseudo-code of the procedure is given in Appendix 5.

In the computation of both Γ and Π, the worst-case complexity of a single insertion in the set Γ (resp. Π) is linear in the size of the set. We can nevertheless prove a worst-case complexity of $O(N2^N)$ in the two cases by remarking that no more than 2^N points can be removed from Γ (resp. Π) in a single run of the algorithm, and that the complexity of a deletion is $O(\log |\Gamma|) = O(N)$ (resp. $O(\log |\Pi|) = O(N)$). This is a worst-case complexity if we assume that the policies are stored in a balanced binary tree. Again, in practice, the pruning procedures *CHTest* and *OPTest* should yield substantial acceleration ratios.

3 Group of data units

We study in this section the complexity of optimal rate-distortion streaming for a group of interdependent data units. Since the rate-distortion optimization problem is NP-hard for a single data unit, it is also NP-hard for a group of data units. We now provide two important results that give relationships between optimal (resp. convex hull) policy vectors and optimal (resp. convex hull) policies.

Lemma 2. *Let $\vec{\pi}^* = (\pi_1^*, \ldots, \pi_L^*)$ be an optimal policy vector. Then all policies $\pi_1^*, \ldots, \pi_L^*$ are optimal.*

Proof. We prove the result by contradiction. Let $\vec{\pi} = (\pi_1, \ldots, \pi_L)$ be a policy vector with at least one non-optimal policy. Let, without loss of generality, π_L be a non-optimal policy, that

is, there exists a policy π' such that $\epsilon(\pi') \leq \epsilon(\pi_L)$ and $\rho(\pi') < \rho(\pi_L)$. Let $\vec{\pi}' = (\pi'_1, \ldots, \pi'_L)$ with $\pi'_1 = \pi_1, \ldots, \pi'_{L-1} = \pi_{L-1}$, $\pi'_L = \pi'$. Then

$$D(\vec{\pi}) = D_0 - \sum_{l=1}^{L} \Delta D_l \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \geq D_0 - \sum_{l=1}^{L} \Delta D_l \prod_{l' \preceq l} (1 - \epsilon(\pi'_{l'})) = D(\vec{\pi}')$$

and

$$R(\vec{\pi}) = \sum_{l=1}^{L} B_l \rho(\pi_l) > \sum_{l=1}^{L} B_l \rho(\pi'_l) = R(\vec{\pi}').$$

Thus, $\vec{\pi}$ cannot be optimal. □

We have a similar result for convex hull policy vectors.

Lemma 3. *Let $\vec{\pi}^* = (\pi_1^*, \ldots, \pi_L^*)$ be a convex hull policy vector. Then all policies $\pi_1^*, \ldots, \pi_L^*$ are convex hull policies.*

Proof. We prove the result by contradiction. Let $\vec{\pi} = (\pi_1, \ldots, \pi_L)$ be a policy vector with at least one policy that is not a convex hull policy. Let, without loss of generality, π_L be one such policy, that is, for all $\lambda' > 0$, there exists a policy π' such that $J_{\lambda'}(\pi') < J_{\lambda'}(\pi_L)$. Then for all $\lambda > 0$

$$\begin{aligned}
J_\lambda(\vec{\pi}) &= D(\vec{\pi}) + \lambda R(\vec{\pi}) \\
&= D_0 + \sum_{l=1}^{L} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\
&= D_0 + \sum_{l=1}^{L-1} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\
&\quad - \Delta D_L \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) + \Delta D_L \epsilon(\pi_L) \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) + \lambda B_L \rho(\pi_L).
\end{aligned}$$

If $\Delta D_L > 0$ and $\epsilon(\pi_{l'}) < 1$ for all $l' \prec L$, then with $\lambda' = \frac{\lambda B_L}{\Delta D_L} \prod_{l' \prec L} \frac{1}{1 - \epsilon(\pi_{l'})}$ and π' such that $J_{\lambda'}(\pi') < J_{\lambda'}(\pi_L)$, we have

$$\begin{aligned}
J_\lambda(\vec{\pi}) &= D_0 + \sum_{l=1}^{L-1} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\
&\quad - \Delta D_L \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) + \Delta D_L J_{\lambda'}(\pi_L) \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) \\
&> D_0 + \sum_{l=1}^{L-1} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\
&\quad - \Delta D_L \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) + \Delta D_L J_{\lambda'}(\pi') \prod_{l' \prec L} (1 - \epsilon(\pi_{l'})) \\
&= J_\lambda(\vec{\pi}'),
\end{aligned}$$

where $\vec{\pi}' = (\pi_1, \ldots, \pi_{L-1}, \pi')$.

If $\Delta D_L = 0$ or if there exists $l' \prec L$ with $\epsilon(\pi_{l'}) = 1$, then

$$\begin{aligned} J_\lambda(\vec{\pi}) &= D_0 + \sum_{l=1}^{L-1} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\ &\quad + \lambda B_L \rho(\pi_L). \end{aligned}$$

Since the convex hull policy $\pi' = (0, \ldots, 0)$ is the only policy whose cost is zero, and since π_L is not a convex hull policy, we have $\rho(\pi_L) > 0$. Thus,

$$\begin{aligned} J_\lambda(\vec{\pi}) &> D_0 + \sum_{l=1}^{L-1} \left[\Delta D_l \left(- \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})) \right) + \lambda B_l \rho(\pi_l) \right] \\ &= J_\lambda(\vec{\pi}'), \end{aligned}$$

where $\vec{\pi}' = (\pi_1, \ldots, \pi_{L-1}, \pi')$. So also in this case $\vec{\pi}$ cannot be a convex hull policy vector. □

3.1 Branch and bound

Lemma 2 allows us to break the problem of finding an optimal policy vector for a group of data units into two parts. First, use the branch and bound algorithm of Section 2.2.2 to compute the set Π of all optimal policies for a single data unit. Then use another branch and bound algorithm, which is similar to the first one, except that instead of having the choice between two branches at each step, we have to choose the next policy among the set Π. We can derive lower bounds for pruning the recursion tree as in the single policy case.

Let $\Pi(i)$ be the ith policy of Π in non-decreasing cost and non-increasing error order. Then $\Pi(0)$ is the policy with the highest error and lowest cost (i.e. $(0, \ldots, 0)$) and $\Pi(|\Pi| - 1)$ is the policy with the lowest error and highest cost (i.e. $(1, \ldots, 1)$).

As in the single data unit case, if we know that policy vector $\vec{\pi}$ begins with prefix $\vec{\pi}' = (\pi'_1, \pi'_2, \ldots, \pi'_{\text{len}(\vec{\pi}')})$, $\pi'_i = \pi_i$ for $i = 1, \ldots, \text{len}(\vec{\pi}')$, $\text{len}(\vec{\pi}') \leq L$, we can give bounds for $D(\vec{\pi})$ and $R(\vec{\pi})$. Since $D(\vec{\pi})$ is smallest if we transmit with the smallest error for each data unit, a lower bound for $D(\vec{\pi})$ is

$$D_{\min}(\vec{\pi}') = D(\pi'_1, \ldots, \pi'_{\text{len}(\vec{\pi}')}, \underbrace{\Pi(|\Pi| - 1), \ldots, \Pi(|\Pi| - 1)}_{L - \text{len}(\vec{\pi}') \text{ times}}). \tag{17}$$

On the other hand, $R(\vec{\pi})$ is smallest if we transmit with the lowest cost for each data unit. Thus, a lower bound for $R(\vec{\pi})$ is

$$R_{\min}(\vec{\pi}') = R(\pi'_1, \ldots, \pi'_{\text{len}(\vec{\pi}')}, \underbrace{\Pi(0), \ldots, \Pi(0)}_{L - \text{len}(\vec{\pi}') \text{ times}}). \tag{18}$$

Thus, a lower bound for the expected Lagrangian $J_\lambda(\vec{\pi})$ is

$$J_{\lambda,\min}(\vec{\pi}') = D_{\min}(\vec{\pi}') + \lambda R_{\min}(\vec{\pi}'). \tag{19}$$

Exploiting Lemmas 2 and 3, and bounds (17), (18), and (19), we can use branch and bound methods similar to the ones used in the single data unit case (see Appendix 6, 7, 8, and 9). Here we define functions *CHTestV* and *OPTestV*, and procedures *CHInsertV* and *OPInsertV* for policy vectors in a similar way to the single policy case.

The same kind of complexity analysis as in the single data unit case can be done, with the difference that the branching factor is not equal to two anymore, but to $|\Pi|$ (resp. $|\Gamma|$) for the optimal policy vectors (resp. convex hull policy vectors). The worst-case complexity of the computation of an optimal policy vector, ignoring all pruning steps, is then $O(N2^N + L^2|\Pi|^L)$ (resp. $O(N2^N + L^2|\Gamma|^L)$).

4 Experimental results

In this section, we provide experimental results to illustrate the improvements allowed by our branch and bound algorithms. As in [2], the probability density function of FTT was modeled as a shifted Gamma distribution with rightward shift κ_F and parameters n_F and α_F [2], that is, for $t \geq \kappa_F$, we have

$$p_F(t) = \frac{\alpha_F}{\Gamma(n_F)}(\alpha_F(t-\kappa_F))^{n_F-1}e^{-\alpha_F(t-\kappa_F)},$$

where Γ is the gamma function. Similarly, the probability density function of BTT was modeled as the shifted Gamma distribution

$$p_B(t) = \frac{\alpha_B}{\Gamma(n_B)}(\alpha_B(t-\kappa_B))^{n_B-1}e^{-\alpha_B(t-\kappa_B)},$$

where κ_B is the rightward shift κ_B and n_B and α_B are parameters.

4.1 Single data unit

We compare the branch and bound algorithm (Appendix 2) to the dynamic programming approach of [2] for computing a policy that minimizes the Lagrangian (3) for a fixed λ. We use the number of visited nodes in the decision process tree as the performance measure. This is a reasonable choice since both algorithms require a similar number of operations per node, with the difference that in the branch and bound algorithm, one more comparison is made to decide if the current branch should be pruned.

In practice, however, we seek the best policy for a given maximum cost, not for a given Lagrange multiplier. With the approach of [2], one must then combine the dynamic programming algorithm with the bisection method to find a convex hull solution for this given cost. A straightforward way to speed up this approach is to replace dynamic programming with our branch and bound technique. However, even with this improvement, this approach has two drawbacks. First, several iterations might be needed until a solution is found. Second, one cannot guarantee that the solution is optimal for the given cost since only convex hull solutions can be found. A better and faster approach is to use the branch and bound algorithm of Section 2.2.2 (Appendix 4). Figure 3 compares the complexity of respectively the dynamic programming approach of [2], the algorithm of Appendix 2 (curves denoted by "Lagrange branch and bound"), and the algorithm of Appendix 4 (curves denoted by "Constrained branch and bound"). In the latter algorithm, the cost constraint was taken as the one corresponding to the solution of the first two algorithms. We provide results for two different Lagrange multipliers. The channel parameters were chosen as in [2]. Figure 4 shows the results for another set of channel parameters.

While dynamic programming must visit all $2^{N+1}-1$ nodes, the results show that the number of visited nodes is much smaller for the two branch and bound algorithms. Note also how the branch and bound algorithm of Appendix 4 was faster than the one of Appendix 2.

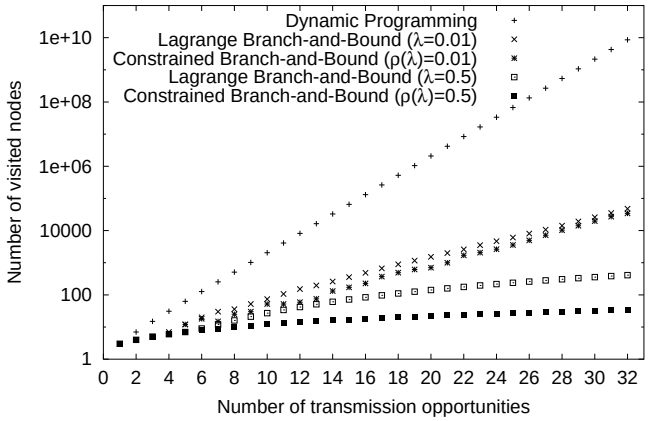

Figure 3: Number of visited nodes as a function of the number of transmission opportunities for the dynamic programming algorithm, the branch and bound algorithm of Appendix 2, and the branch and bound algorithm of Appendix 4. The results are shown for Lagrange multiplier $\lambda = 0.01$ and 0.5. The channel parameters are $\epsilon_F = \epsilon_B = 0.2, \kappa_F = \kappa_B = 25$ ms $, n_F = n_B = 2, 1/\alpha_F = 1/\alpha_B = 12.5$. The number of transmission opportunities is $N = 8$, the time interval between two opportunities is 50 ms.

4.2 Group of data units

The branch and bound algorithms for computing optimal policy vectors are too time-consuming for online applications. But they can be used in offline applications or to evaluate the quality of heuristic solutions. We now compare the quality of the solutions found by the SA algorithm to the ones computed with the branch and bound algorithm for a set of data units and a given rate constraint (Appendix 8). We provide two examples that show that the SA algorithm can perform poorly. As in [2], the stopping criterion for the SA algorithm is $J_\lambda(\vec{\pi}^{(k-1)}) = J_\lambda(\vec{\pi}^{(k)})$, and the single policies are changed in round-robin style ($l_k = ((k-1) \bmod L) + 1$).

We encoded the Foreman video sequence in CIF size according to the Video-CD standard (MPEG1, 1150 kilobits per second with a frame rate of 25 frames per second) and computed policy vectors for a group of ten frames (frame 13 to frame 22). The frame sequence was IBBPBBPBBP, where I-frames are independent, P-frames depend on the last P- or I-frame, and B-frames depend on the last and next P- or I-frame. The frame sizes (in bits) were $(B_1, \ldots, B_{10})$=(211048, 30252, 24996, 178508, 20820, 20292, 81988, 26820, 15164, 77676). The distortions in dB were $(D_0, \Delta D_1, \ldots, \Delta D_{10})$ =(11.78, 3.35, 3.01, 3.06, 3.53, 2.94, 2.93, 3.26, 2.98, 3.08, 3.24). They were computed as follows. Let X_i, $i = 1, \ldots, 10$ denote an original frame, Y_i, $i = 1, \ldots, 10$ denote a decoded frame, G denote a constant frame where all pixel values are equal to 128. Then $D_0 = \frac{1}{10}\sum_{i=1}^{10} PSNR(X_i, G)$ and $\Delta D_i = \frac{1}{10}(PSNR(X_i, Y_i) - PSNR(X_i, G))$, where PSNR is the peak-signal-to-noise-ratio of the Y component. Finally, the number of transmission opportunities was $N = 8$, and the channel parameters were as in Fig. 3.

The SA solution for Lagrange multiplier $\lambda = 6.4 \cdot 10^{-5}$ was ((1, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 1, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 1, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 1, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0)), yielding an expected rate of 756,566 bits and an expected PSNR of 29.97 dB. The branch and bound algorithm was able to find a much better solution. Indeed, for rate constraint 756,560 bits, it found the solution ((1, 0, 0, 0, 1, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1,

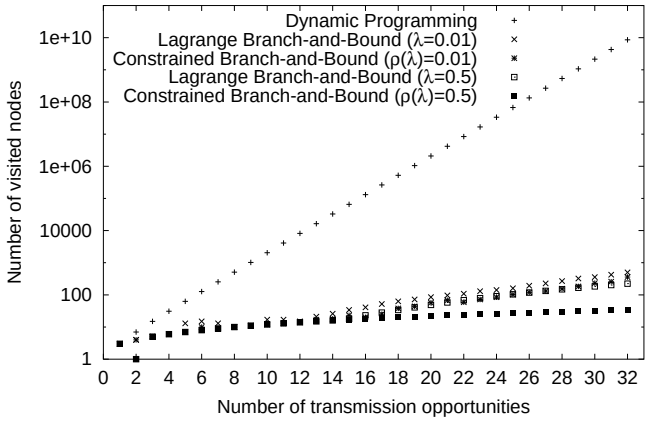

Figure 4: Number of visited nodes as a function of the number of transmission opportunities for the dynamic programming algorithm, the branch and bound algorithm of Appendix 2, and the branch and bound algorithm of Appendix 4. The results are shown for Lagrange multiplier $\lambda = 0.01$ and 0.5. The channel parameters are $\epsilon_F = \epsilon_B = 0.01, \kappa_F = \kappa_B = 25$ ms $, n_F = n_B = 8, 1/\alpha_F = 1/\alpha_B = 12.5$. The number of transmission opportunities is $N = 8$, the time interval between two opportunities is 50 ms.

0, 0, 0, 1, 0, 0, 0), (1, 0, 0, 0, 1, 0, 0, 0), (1, 0, 0, 1, 0, 0, 1, 0), (1, 0, 0, 0, 1, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0)), whose expected rate and PSNR are 756,560 bits and 30.67 dB, respectively.

For $\lambda = 7.2 \cdot 10^{-5}$ and the same channel parameters, the SA solution was ((0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (1, 0, 0 ,0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0)), yielding an expected rate of 341,768 bits and an expected PSNR of 11.78 dB. For this rate constraint, the branch and bound algorithm found solution (1, 0, 0, 1, 0, 1, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0)), whose expected rate and PSNR were 341,187 bits and 15.10 dB, respectively.

Note that in the last case, the SA solution transmits B- and P- frames without sending the I-frame they depend on, which is not a reasonable strategy.

5 Conclusion

This work has three important contributions. First, it showed that the problem of optimal rate-distortion streaming of packetized multimedia in the sender-driven transmission over a single QoS network using retransmissions with feedback is NP-hard. This justifies the use of fast heuristic techniques such as the SA algorithm. Second, it gives a branch and bound algorithm for minimizing the Lagrangian for a single data unit. Our algorithm is much faster than the dynamic programming approach of [2]. Thus, it can be used to speed up the SA algorithm. Finally, the work introduces branch and bound algorithms for computing optimal policies for a group of data units. These algorithms can be used in offline applications or to evaluate the quality of suboptimal solutions.

APPENDIX 1 Proof of equality (2)

From the definition of the mean, $\rho(\pi)$ is equal to the sum for each i of terms of the form $\sum_{j=0}^{i} \pi(j)$ multiplied by the probability that the first acknowledgment is received in the time interval $]s_i, s_{i+1}]$. For convenience, we assume $s_{\mathrm{len}(\pi)} = +\infty$ and $P\{\overline{RTT} > 0\} = 1$. The probability that the first acknowledgment is received between times s_i and s_{i+1} can be written

$$\left(\prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_i - s_j\} \right) \cdot \left(1 - \prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_{i+1} - s_j \mid \overline{RTT} > s_i - s_j\} \right),$$

where the first factor is the probability that no acknowledgment was received at time s_i, and the second factor is the probability that an acknowledgment is received at time s_{i+1} given that no acknowledgment was received at time s_i. We denote by PN_i the first factor and by $1 - PC_i$ the second factor.

Hence

$$\begin{aligned}
\rho(\pi) &= \sum_{i=0}^{\mathrm{len}(\pi)-1} \left(\sum_{j=0}^{i} \pi(j) \right) PN_i \cdot (1 - PC_i) \\
&= \sum_{i=0}^{\mathrm{len}(\pi)-1} \left(\sum_{j=0}^{i} \pi(j) \right) (PN_i - PN_i \cdot PC_i) \\
&= \sum_{i=0}^{\mathrm{len}(\pi)-1} \left(\sum_{j=0}^{i} \pi(j) \right) \left[\prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_i - s_j\} - \prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_{i+1} - s_j\} \right]
\end{aligned}$$

The last line is due to

$$P\{\overline{RTT} > s_{i+1} - s_j \mid \overline{RTT} > s_i - s_j\} = \frac{P\{\overline{RTT} > s_{i+1} - s_j\}}{P\{\overline{RTT} > s_i - s_j\}}.$$

Now we can separate into two distinct sums

$$\begin{aligned}
\rho(\pi) &= \left(\sum_{i=0}^{\mathrm{len}(\pi)-1} \left(\sum_{j=0}^{i} \pi(j) \right) \prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_i - s_j\} \right) \\
&\quad + \left(\sum_{i=0}^{\mathrm{len}(\pi)-1} \left(\sum_{j=0}^{i} \pi(j) \right) \prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_{i+1} - s_j\} \right)
\end{aligned}$$

If we change the summation index and let $k = i + 1$, we have for the second sum

$$\sum_{k=1}^{\mathrm{len}(\pi)} \left(\sum_{j=0}^{k-1} \pi(j) \right) \prod_{\substack{j<k \\ \pi(j)=1}} P\{\overline{RTT} > s_k - s_j\}.$$

By summing the corresponding terms in the two sums, we obtain

$$\begin{aligned}\rho(\pi) &= \pi(0) \\ &+ \sum_{i=1}^{\mathrm{len}(\pi)-1} \left(\prod_{\substack{j<i \\ \pi(j)=1}} P\{\overline{RTT} > s_i - s_j\} \right) \left[\left(\sum_{j=0}^{i} \pi(j) \right) - \left(\sum_{j=0}^{i-1} \pi(j) \right) \right] \\ &+ \left(\sum_{j=0}^{\mathrm{len}(\pi)-1} \pi(j) \right) \left(\prod_{\substack{j \le \mathrm{len}(\pi)-1 \\ \pi(j)=1}} P\{\overline{RTT} > s_{\mathrm{len}(\pi)} - s_j\} \right).\end{aligned}$$

The last line is equal to zero from our assumption that $s_{\mathrm{len}(\pi)} = +\infty$. The quantity in brackets in the second line is equal to $\pi(i)$. Hence we obtain the announced result.

APPENDIX 2
Lagrange optimization for a single policy

$\pi \Leftarrow (0, \ldots, 0)$
$\pi' \Leftarrow ()$
BBLagrangeOptimizationRecursion(π')
end

Recursive procedure **BBLagrangeOptimizationRecursion(π')**:
$\pi'_0 \Leftarrow (\pi'(0), \ldots, \pi'(\mathrm{len}(\pi') - 1), 0)$
$\pi'_1 \Leftarrow (\pi'(0), \ldots, \pi'(\mathrm{len}(\pi') - 1), 1)$
$J_0 \Leftarrow J_{\lambda,\min}(\pi'_0)$
$J_1 \Leftarrow J_{\lambda,\min}(\pi'_1)$
if $J_0 \le J_1$ **then**
  if $J_0 < J_\lambda(\pi)$ **then**
    if $\mathrm{len}(\pi'_0) = \mathrm{len}(\pi)$ **then**
      $\pi \Leftarrow \pi'_0$
    else
      BBLagrangeOptimizationRecursion(π'_0)
    end if
  end if
  if $J_1 < J_\lambda(\pi)$ **then**
    if $\mathrm{len}(\pi'_1) = \mathrm{len}(\pi)$ **then**
      $\pi \Leftarrow \pi'_1$
    else
      BBLagrangeOptimizationRecursion(π'_1)
    end if
  end if
else
  if $J_1 < J_\lambda(\pi)$ **then**
    if $\mathrm{len}(\pi'_1) = \mathrm{len}(\pi)$ **then**
      $\pi \Leftarrow \pi'_1$
    else
      BBLagrangeOptimizationRecursion(π'_1)

```
    end if
  end if
  if J_0 < J_λ(π) then
    if len(π'_0) = len(π) then
      π ⇐ π'
    else
      BBLagrangeOptimizationRecursion(π'_0)
    end if
  end if
end if
```

APPENDIX 3
Computation of all convex hull policies

At termination, the convex hull policies are in Γ.

```
Γ ⇐ {(0,...,0),(1,...,1)}
π' ⇐ (1)
BBConvexHullRecursion(π')
end
```

Recursive procedure **BBConvexHullRecursion(π')**:

```
Increase length of π' by 1
π'(len(π') − 1) ⇐ 0
if CHTest(Γ,π')="yes" then
  if len(π') = l then
    CHInsert(Γ,π')
  else
    BBConvexHullRecursion(π')
  end if
end if
π'(len(π') − 1) ⇐ 1
if CHTest(Γ,π')="yes" then
  if len(π') = l then
    CHInsert(Γ,π')
  else
    BBConvexHullRecursion(π')
  end if
end if
```

APPENDIX 4
Optimization for a single policy and a rate constraint

```
π ⇐ (0,...,0)
π' ⇐ ()
BBConstrainedOptimizationRecursion(π')
end
```

Recursive procedure **BBConstrainedOptimizationRecursion(π')**:

```
π'_0 ⇐ (π'(0),...,π'(len(π') − 1),0)
```

$\pi'_1 \Leftarrow (\pi'(0), \ldots, \pi'(\text{len}(\pi') - 1), 1)$
$\epsilon_0 \Leftarrow \epsilon_{\min}(\pi'_0)$
$\epsilon_1 \Leftarrow \epsilon_{\min}(\pi'_1)$
if $\epsilon_0 \leq \epsilon_1$ **then**
  if $\epsilon_0 \leq \epsilon(\pi)$ and $\rho_{\min}(\pi_0) \leq \rho_{\max}$ **then**
    if $\text{len}(\pi'_0) = \text{len}(\pi)$ **then**
      if $\epsilon_0 < \epsilon(\pi)$ or $(\epsilon_0 = \epsilon(\pi)$ and $\rho_{\min}(\pi_0) < \rho(\pi))$ **then**
        $\pi \Leftarrow \pi'_0$
      end if
    else
      BBConstrainedOptimizationRecursion(π'_0)
    end if
  end if
  if $\epsilon_1 \leq \epsilon(\pi)$ and $\rho_{\min}(\pi_1) \leq \rho_{\max}$ **then**
    if $\text{len}(\pi'_1) = \text{len}(\pi)$ **then**
      if $\epsilon_1 < \epsilon(\pi)$ or $(\epsilon_1 = \epsilon(\pi)$ and $\rho_{\min}(\pi_1) < \rho(\pi))$ **then**
        $\pi \Leftarrow \pi'_1$
      end if
    else
      BBConstrainedOptimizationRecursion(π'_1)
    end if
  end if
else
  if $\epsilon_1 \leq \epsilon(\pi)$ and $\rho_{\min}(\pi_1) \leq \rho_{\max}$ **then**
    if $\text{len}(\pi'_1) = \text{len}(\pi)$ **then**
      if $\epsilon_1 < \epsilon(\pi)$ or $(\epsilon_1 = \epsilon(\pi)$ and $\rho_{\min}(\pi_1) < \rho(\pi))$ **then**
        $\pi \Leftarrow \pi'_1$
      end if
    else
      BBConstrainedOptimizationRecursion(π'_1)
    end if
  end if
  if $\epsilon_0 \leq \epsilon(\pi)$ and $\rho_{\min}(\pi_0) \leq \rho_{\max}$ **then**
    if $\text{len}(\pi'_0) = \text{len}(\pi)$ **then**
      if $\epsilon_0 < \epsilon(\pi)$ or $(\epsilon_0 = \epsilon(\pi)$ and $\rho_{\min}(\pi_0) < \rho(\pi))$ **then**
        $\pi \Leftarrow \pi'_0$
      end if
    else
      BBConstrainedOptimizationRecursion(π'_0)
    end if
  end if
end if

APPENDIX 5
Computation of all optimal policies

At termination, the optimal policies are in Π.

$\Pi \Leftarrow \{(0, \ldots, 0), (1, \ldots, 1)\}$
$\pi' \Leftarrow (1)$

BBOptimalPointsRecursion(π')
end

Recursive procedure **BBOptimalPointsRecursion(π')**:
Increase length of π' by 1
$\pi'(\text{len}(\pi') - 1) \Leftarrow 0$
if OPTest(Π,π')="yes" **then**
  if len(π') = l **then**
    OPInsert(Π,π')
  else
    BBOptimalPointsRecursion(π')
  end if
end if
$\pi'(\text{len}(\pi') - 1) \Leftarrow 1$
if OPTest(Π,π')="yes" **then**
  if len(π') = l **then**
    OPInsert(Γ,π')
  else
    BBOptimalPointsRecursion(π')
  end if
end if

APPENDIX 6
Lagrange optimization for a policy vector

$\vec{\pi} \Leftarrow$ Solution of SA algorithm
$\vec{\pi}' \Leftarrow ()$
BBGOPLagrangeOptimizationRecursion($\vec{\pi}'$)
end

Recursive procedure **BBGOPLagrangeOptimizationRecursion($\vec{\pi}'$)**:
if len($\vec{\pi}'$) = L **then**
  $\vec{\pi} \Leftarrow \vec{\pi}'$
else
  for $i = 0$ to $|\Gamma| - 1$ **do**
    $\vec{\pi}'_i \Leftarrow (\vec{\pi}'(0), \ldots, \vec{\pi}'(\text{len}(\vec{\pi}') - 1), \Gamma(i))$
    if $J_{\lambda,\min}(\vec{\pi}'_i) < J_\lambda(\vec{\pi})$ **then**
      Insert i into a set I
    end if
  end for
  Sort I such that $J_{\lambda,\min}(\vec{\pi}'_{I(k)}) \leq J_{\lambda,\min}(\vec{\pi}'_{I(k+1)})$ $\forall k \in \{0, \ldots, |I| - 2\}$
  for $k = 0$ to $|I| - 1$ **do**
    if $J_{\lambda,\min}(\vec{\pi}'_{I(k)}) < J_\lambda(\vec{\pi})$ **then**
      BBGOPLagrangeOptimizationRecursion($\vec{\pi}'_{I(k)}$)
    end if
  end for
end if

APPENDIX 7
Computation of all convex hull policy vectors

At termination, the convex hull policy vectors are in $\vec{\Gamma}$.
$\vec{\Gamma} \Leftarrow \{(\Gamma(0), \ldots, \Gamma(0)), (\Gamma(|\Gamma|-1), \ldots, \Gamma(|\Gamma|-1))\}$
$\vec{\pi}' \Leftarrow ()$
GOPBBConvexHullRecursion($\vec{\pi}'$)
end

Recursive procedure **GOPBBConvexHullRecursion($\vec{\pi}'$)**:
if len($\vec{\pi}'$) $= L$ **then**
  CHInsertV($\vec{\Gamma}$,$\vec{\pi}'$)
else
  for $i = 0$ to $|\Gamma| - 1$ **do**
    $\vec{\pi}'_i \Leftarrow (\vec{\pi}'(0), \ldots, \vec{\pi}'(\text{len}(\vec{\pi}') - 1), \Gamma(i))$
    if CHTestV($\vec{\Gamma}$,$\vec{\pi}'_i$)="yes" **then**
      Insert i into a set I
    end if
  end for
  Sort I such that $\frac{D_{\min}(\vec{\pi}'_{I(k)})}{R_{\min}(\vec{\pi}'_{I(k)})} \geq \frac{D_{\min}(\vec{\pi}'_{I(k+1)})}{R_{\min}(\vec{\pi}'_{I(k+1)})}$ $\forall k \in \{0, \ldots, |I| - 2\}$
  for $k = 0$ to $|I| - 1$ **do**
    if CHTestV($\vec{\Gamma}$,$\vec{\pi}'_{I(k)}$)="yes" **then**
      BBGOPConvexHullRecursion($\vec{\pi}'_{I(k)}$)
    end if
  end for
end if

APPENDIX 8
Computation of an optimal policy vector for a rate constraint

$\vec{\pi} \Leftarrow (\Pi(0), \ldots, \Pi(0))$
$\vec{\pi}' \Leftarrow ()$
BBGOPConstrainedOptimizationRecursion($\vec{\pi}'$)
end

Recursive procedure **BBGOPConstrainedOptimizationRecursion($\vec{\pi}'$)**:
if len($\vec{\pi}'$) $= L$ **then**
  if $D_{\min}(\vec{\pi}') < D(\vec{\pi})$ or ($D_{\min}(\vec{\pi}') = D(\vec{\pi})$ and $R_{\min}(\vec{\pi}') < R_\pi$) **then**
    $\vec{\pi} \Leftarrow \vec{\pi}'$
  end if
else
  for $i = 0$ to $|\Pi| - 1$ **do**
    $\vec{\pi}'_i \Leftarrow (\vec{\pi}'(0), \ldots, \vec{\pi}'(\text{len}(\vec{\pi}') - 1), \Pi(i))$
    if $D_{\min}(\vec{\pi}'_i) \leq D(\vec{\pi})$ and $R_{\min}(\vec{\pi}'_i) \leq R_{\max}$ **then**
      Insert i into a set I
    end if
  end for
  Sort I such that $D_{\min}(\vec{\pi}'_{I(k)}) \leq D_{\min}(\vec{\pi}'_{I(k+1)})$ $\forall k \in \{0, \ldots, |I| - 2\}$
  for $k = 0$ to $|I| - 1$ **do**
    if $D_{\min}(\vec{\pi}'_{I(k)}) \leq D(\vec{\pi})$ **then**
      BBGOPConstrainedOptimizationRecursion($\vec{\pi}'_{I(k)}$)

end if
end for
end if

APPENDIX 9
Computation of all optimal policy vectors

At termination, the optimal policy vectors are in $\vec{\Pi}$.
$\vec{\Pi} \Leftarrow \{(\Pi(0), \ldots, \Pi(0)), (\Pi(|\Pi|-1), \ldots, \Pi(|\Pi|-1))\}$
$\vec{\pi}' \Leftarrow ()$
GOPBBOptimalPointsRecursion($\vec{\pi}'$)
end

Recursive procedure **GOPBBOptimalPointsRecursion($\vec{\pi}'$)**:
if $\text{len}(\vec{\pi}') = L$ **then**
OPInsertV($\vec{\Pi}$,$\vec{\pi}'$)
else
for $i = 0$ to $|\Pi| - 1$ **do**
$\vec{\pi}'_i \Leftarrow (\vec{\pi}'(0), \ldots, \vec{\pi}'(\text{len}(\vec{\pi}')-1), \Pi(i))$
if OPTestV($\vec{\Pi}$,$\vec{\pi}'_i$)="yes" **then**
Insert i into a set I
end if
end for
Sort I such that $\frac{D_{\min}(\vec{\pi}'_{I(k)})}{R_{\min}(\vec{\pi}'_{I(k)})} \geq \frac{D_{\min}(\vec{\pi}'_{I(k+1)})}{R_{\min}(\vec{\pi}'_{I(k+1)})}$ $\forall k \in \{0, \ldots, |I|-2\}$
for $k = 0$ to $|I| - 1$ **do**
if OPTestV($\vec{\Pi}$,$\vec{\pi}'_{I(k)}$)="yes" **then**
BBGOPOptimalPointsRecursion($\vec{\pi}'_{I(k)}$)
end if
end for
end if

Acknowledgments. We thank Phil Chou for helpful discussions.

References

[1] P. A. Chou and Z. Miao, *Rate-distortion optimized streaming of packetized media*, Microsoft Research Technical Report MSR-TR-2001-35, Feb. 2001.

[2] P. A. Chou and Z. Miao, *Rate-distortion optimized streaming of packetized media*, *IEEE Transactions on Multimedia*, submitted Feb. 2001.

[3] J. Chakareski, P. A. Chou, and B. Aazhang, *Computing rate-distortion optimized policies for streaming media to wireless clients*, *Proc. DCC'02*, Utah, April 2002.

[4] J. Chakareski and B. Girod, *Server diversity in rate-distortion optimized media streaming*, *Proc. IEEE ICIP'03*, Barcelona, Sept. 2003.

[5] M. Kalman, P. Ramanathan, and B. Girod, *Rate-Distortion optimized video streaming with multiple deadlines*, *Proc. IEEE ICIP'03*, Barcelona, Sept. 2003.

[6] G. Cheung and C. Chan, *Jointly optimal reference frame & quality of service selection for H.26L video coding over lossy networks*, *Proc. IEEE ICME'03*, Baltimore, July 2003.

[7] F.P. Preparata and M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability,* W.H. Freeman and Company, New York, 1979.

[9] D. Pisinger, *Algorithms for Knapsack Problems*, PhD thesis, University of Copenhagen, 1995.

[10] R. Szekli, *Stochastic Ordering and Dependence in Applied Probability*, Lecture Notes in Statistics 97, Springer-Verlag, 1995.