Efficient rate-distortion optimized media streaming for tree-reducible packet dependencies

Martin Röder[†a], Jean Cardinal[‡b], Raouf Hamzaoui[*a]

[a]Department of Computer and Information Science, University of Konstanz, Germany
[b]Computer Science Department, Université Libre de Bruxelles, Belgium

ABSTRACT

In packetized media streaming systems, packet dependencies are often modeled as a directed acyclic graph called the dependency graph. We consider the situation where the dependency graph is reducible to a tree. This occurs, for instance, in MPEG1 video streams that are packetized at the frame level. Other video coding standards such as H.264 also allow tree-reducible dependencies. We propose in this context efficient dynamic programming algorithms for finding rate-distortion optimal transmission policies. The proposed algorithms are much faster than previous exact algorithms developed for arbitrary dependency graphs.

1. INTRODUCTION

Multimedia streaming, in particular video streaming, is one of the most popular multimedia services over the internet. It allows the receiver to continuously play back a compressed multimedia stream while parts of it are still being received. Streaming systems must be able to deal with transmission errors and bandwidth limitations in a flexible way, and the design of efficient algorithms for finding optimal transmission strategies is a challenging question.

In this paper, we concentrate on packetized media streams, in which the compressed bitstream is already divided into packets. We propose algorithms for finding rate-distortion optimal transmission policies, i.e., transmission strategies that optimally trade bits for reconstruction quality. We use the popular framework introduced by Chou and Miao [1], a flexible model for rate-distortion optimized streaming systems over packet erasure channels. This framework has been recognized as one of the most important recent contribution in multimedia streaming [2].

In this framework, the dependencies between multiple packets are modeled as a directed acyclic graph, in which there is an edge from packet i to packet i' whenever packet i' cannot be decoded if packet i is not decoded. This dependency model makes the framework suitable to a wide range of source coders, including the new video standard H.264 [3]. Each single packet can be transmitted at a finite number of transmission opportunities. In a sender-driven scenario, the receiver sends an acknowledgment each time a packet is received before its delivery deadline. As long as no acknowledgment is received and the deadline is not reached, the sender is allowed to retransmit the packet according to a transmission policy. A multiple-packet transmission policy is a combination of single-packet transmission policies. The SA algorithm [1] is a fast heuristic technique that optimizes multiple-packet transmission policies with respect to a rate-distortion criterion. It breaks the complexity of the problem by optimizing for one single-packet policy at a time.

Numerous contributions have already been proposed within this framework. For example, it was adapted for a wireless channel [4] and extended to include multiple delivery deadlines [5]. Other important contributions include a novel acknowledgment mechanism [6] and a very fast method for computing a transmission strategy for a group of packets using an approximated distortion model [7].

In [8], we devised branch and bound algorithms for computing both single- and multiple-packet transmission policies. The algorithm for single-packet policies outperformed the original optimization algorithm proposed by Chou and Miao [1] by a significant margin. It is also shown in the same paper that finding optimal policies is an

[†]roeder@inf.uni-konstanz.de, [‡]jcardin@ulb.ac.be, [*]hamzaoui@inf.uni-konstanz.de. This work was supported by the DFG Research Training Group GK-1042 "Explorative Analysis and Visualization of large Information Spaces".

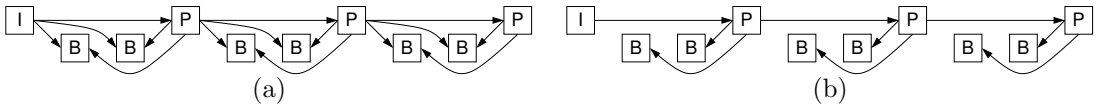


Figure 1. (a) Dependency graph for an MPEG-1 group of frames (for clarity, not all redundant edges are shown), (b) Transitive reduction of (a).

NP-hard problem. More recently, we proposed a dynamic programming algorithm [9] that provides optimal single-packet transmission policies if the forward- and round-trip times of the packets have exponential distributions.

The main contribution of this paper is the important observation that rate-distortion optimal policies for a group of interdependent packets can be efficiently computed in the special case where the transitive reduction of the dependency graph, obtained by removing redundant edges, is a tree. For example, an MPEG-1 encoded video stream consists of I-, P-, and B-frames. I-frames do not depend on other frames, P-frames depend on the preceeding I- or P-frame, and B-frames depend on the preceeding I- or P-frame and on the following I- or P-frame. However, the dependence of a B-frame on its preceeding I- or P-frame is redundant, since the P-frame following the B-frame also depends on the I- or P-frame preceeding the B-frame (Fig. 1). So the transitive reduction of the dependency graph of any MPEG1 encoded video stream that is packetized at frame level is a set of independent trees, where each tree corresponds to one group of frames.

The next section contains useful definitions and notations that are used in the paper. In Section 3 and 4, we propose dynamic programming algorithms for finding optimal and convex hull policies for multiple packets when the dependency graph is tree reducible. In section 5, we propose thinning methods for reducing the complexity of these algorithms at the cost of a controllable increase in distortion. In Section 6, we describe experimental results, showing that the proposed algorithms are faster than the previous algorithms of [8] and provide better solutions than those of the SA algorithm [1].

2. PRELIMINARIES

This section presents the terminology and notations used throughout the paper. The definitions related to tranmission policies are essentially those of the previous works [1,8]. Definitions and notations related to dependency graphs are mostly standard graph-theoretic ones.

We consider the problem of transmitting a set of interdependent packets in a rate-distortion optimized way [1]. The dependency between the packets is modeled with a directed acyclic graph (DAG). Let $G = (V, E)$ be a DAG where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. If there is a path in G from a vertex i' to a vertex i, we write $i' \prec_G i$ and say that i' is an *ancestor* of i in G. We also use the notation $i' \preceq_G i$ if $i' \prec_G i$ or $i' = i$. Note that the ancestor relation is transitive. If $i'' \prec_G i'$ and $i' \prec_G i$, then $i'' \prec_G i$. We refer to the set of vertices of a DAG G as $V(G)$ and to the set of edges as $E(G)$. The *in-degree* of a vertex $i \in V(G)$ is the number of edges that end in i. The *out-degree* of i is the number of edges that start in i. The transitive reduction of a DAG G is the DAG G^t with the smallest number of edges such that $(G^t)^T = G^T$ where G^T is the transitive closure of G [10]. The *dependency graph* of a set $V = \{1, \ldots, L\}$ of L interdependent packets is the DAG $G = (V, E)$ where the set of edges E is given by $(i', i) \in E$ if packet i' must be decoded so that packet i can also be decoded. An example of the transitive reduction of a dependency graph is shown in Fig. 1. Note that G^t is uniquely determined by G and that $V(G^t) = V(G)$. G^t can be computed in $O(|V(G)||E(G)|)$ time [11]. We call an edge (i', i) of a DAG G *redundant* if $(i', i) \notin E(G^t)$. We define the union of two induced subgraphs G_1 and G_2 of a DAG G, $G_1 \cup_G G_2$, as the induced subgraph of G with $V(G_1 \cup_G G_2) = V(G_1) \cup V(G_2)$. Two subgraphs G_1 and G_2 of a DAG G are said to be *independent* in G if there is no edge in G that connects a vertex of G_1 to a vertex of G_2 or vice versa. If G_1 and G_2 are induced subgraphs of a DAG G and independent in G, then $G_1 \cup_G G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$.

A DAG T where exactly one vertex $r \in V(T)$ has an in-degree of 0 and all other vertices have an in-degree of 1 is called a tree with root r. We also denote the root of a tree T by $r(T)$, and we call the subgraph $(\{r(T)\}, \emptyset)$ of T the *root graph* of T. Any vertex of T with an out-degree of 0 is called a leaf of T. We refer to the set of leaves of T as $L(T)$. A subgraph T' of T which is a tree with $L(T') \subseteq L(T)$ is called a subtree of T. A subtree T' of T that contains all vertices of T that have $r(T')$ as an ancestor is called a *full subtree* of T. All full subtrees T'

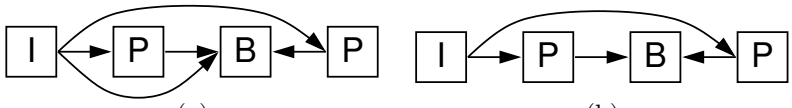


Figure 2. (a) Dependency graph. (b) Transitive reduction of (a). (a) is not tree reducible.

of T which have a root that is adjacent to $r(T)$ in T, that is, $(r(T), r(T')) \in E(T)$, are called *direct subtrees* of T. Note that two direct subtrees of a tree T are always independent in T. A *subforest* of T is either a subtree of T or the union of two or more subtrees of T. A *direct subforest* of T is either a direct subtree of T or the union of two or more direct subtrees of T. The union of all direct subtrees of T is called the *complete subforest* of T. A graph whose transitive reduction is a tree is said to be tree reducible. For example, the dependency graph of Fig. 1(a) is tree reducible, whereas the dependency graph of Fig. 2(a) is not. The transitive reduction of a tree reducible dependency graph G is called the *dependency tree* of G.

The transmission of a single packet is controlled by a policy π. Each transmission policy π has an error $\epsilon(\pi)$, $0 \leq \epsilon(\pi) \leq 1$, which is the probability that a packet transmitted with policy π does not arrive at the receiver on time, and a cost $\rho(\pi)$, $\rho(\pi) \geq 0$, which is the expected number of transmissions of a packet transmitted with policy π. The transmission policy, either sender- or receiver-driven, can be for instance defined by a bit vector representing the transmissions times, as in [1] and [8], or can be any other transmission strategy (e.g., [12]) yielding the guarantees $\epsilon(\pi)$ and $\rho(\pi)$ on the loss probability and average retransmission times, respectively. A policy π^* is said to be an *optimal policy* if there exists no other policy π such that $\epsilon(\pi) \leq \epsilon(\pi^*)$ and $\rho(\pi) < \rho(\pi^*)$. A policy π^* is said to be a *convex hull policy* if there exists $\lambda \geq 0$ such that π^* minimizes the Lagrangian $\epsilon(\pi) + \lambda\rho(\pi)$ for all policies π. All convex hull policies are optimal, but not all optimal policies are convex hull policies.

Let $V = \{1, \ldots, L\}$ be a set of packets and G be a DAG with $V(G) = V$. A policy vector $\vec{\pi} = (\pi_1, \ldots, \pi_L)$ for G is a vector of transmission policies, where π_i is the policy for packet $i \in V$. Let G' be a subgraph of G with $V(G') = \{s_1, \ldots, s_k\} \subseteq V(G)$. The policy vector $\vec{\pi}' = (\pi_{s_1}, \ldots, \pi_{s_k})$, $1 \leq s_1 < \ldots < s_k \leq L$, is called the *policy subvector* of $\vec{\pi}$ for G'. We denote by B_i the size of packet i and by ΔD_i the expected reduction in reconstruction error if packet i is decoded on time. The expected rate of a policy vector $\vec{\pi}$ with respect to the DAG G is

$$R_G(\vec{\pi}) = \sum_{i \in V(G)} B_i \rho(\pi_i) \tag{1}$$

and the expected reconstruction error reduction is

$$D_G(\vec{\pi}) = -\sum_{i \in V(G)} \Delta D_i \prod_{i' \preceq_G i} (1 - \epsilon(\pi_{i'})). \tag{2}$$

Let D_0 be the distortion for the set V if no packet is received. Then the expected distortion of the transmission of V with policy vector $\vec{\pi}$ is

$$\bar{D}_G(\vec{\pi}) = D_0 + D_G(\vec{\pi}).$$

A policy vector $\vec{\pi}^*$ is *optimal* for a DAG G if there exists no policy vector $\vec{\pi}$ such that $D_G(\vec{\pi}) \leq D_G(\vec{\pi}^*)$ and $R_G(\vec{\pi}) < R_G(\vec{\pi}^*)$. A policy vector $\vec{\pi}^*$ is a *convex hull policy vector* for a DAG G if there exists $\lambda \geq 0$ such that $\vec{\pi}^*$ minimizes the Lagrangian

$$J_{G,\lambda}(\vec{\pi}) = D_G(\vec{\pi}) + \lambda R_G(\vec{\pi}).$$

It is easy to see that the expected rate and the expected reconstruction error reduction of a policy with a tree reducible dependency graph are the same as with its dependency tree.

PROPOSITION 1. *Let G be a tree-reducible dependency graph and T be the dependency tree of G. Then for any policy vector π for G, we have $R_G(\vec{\pi}) = R_T(\vec{\pi})$ and $D_G(\vec{\pi}) = D_T(\vec{\pi})$.*

Proof. Since T is the transitive reduction of G, we have $G^T = T^T$ and $V(G) = V(T)$, so $i' \preceq_G i$ if and only if $i' \preceq_T i$. The proof follows then from (1) and (2). □

Proposition 1 allows us to restrict the study of tree-reducible dependency graphs to their dependency trees.

We now give some important properties of policy vectors for dependency trees. Let G_1 and G_2 be two induced subgraphs of a DAG G such that $V(G_1) \cap V(G_2) = \emptyset$. Let $\vec{\pi}'$ and $\vec{\pi}''$ be policy vectors for G_1 and G_2, respectively. Thus, if $V(G_1) = \{s_1, \ldots, s_k\}$ and $V(G_2) = \{t_1, \ldots, t_m\}$, we have $\vec{\pi}' = (\pi'_{s_1}, \ldots, \pi'_{s_k})$ and $\vec{\pi}'' = (\pi''_{t_1}, \ldots, \pi''_{t_m})$. We define the product $\vec{\pi}' \times \vec{\pi}''$ as the policy vector for $G_1 \cup_G G_2$ that contains all policies of $\vec{\pi}'$ and $\vec{\pi}''$, ordered according to the labels of their associated packets. That is, $\vec{\pi}' \times \vec{\pi}'' = (\pi_{v_1}, \ldots, \pi_{v_{k+m}})$, where $v_1 < v_2 < \ldots < v_{k+m}$, $v_j \in V(G_1) \cup V(G_2)$, and

$$\pi_{v_j} = \begin{cases} \pi'_{v_j} \text{ if } v_j \in V(G_1) \\ \pi''_{v_j} \text{ if } v_j \in V(G_2) \end{cases}$$

for $1 \leq j \leq k+m$. It is easy to show that

$$R_{G_1 \cup_G G_2}(\vec{\pi}' \times \vec{\pi}'') = R_{G_1}(\vec{\pi}') + R_{G_2}(\vec{\pi}''). \tag{3}$$

In general, $D_{G_1 \cup_G G_2}(\vec{\pi}' \times \vec{\pi}'')$ cannot be expressed as easily. However, if T_1 and T_2 are subforests of a tree T and independent in T, and $\vec{\pi}'$ and $\vec{\pi}''$ are policy vectors for T_1 and T_2, respectively, then

$$D_{T_1 \cup_T T_2}(\vec{\pi}' \times \vec{\pi}'') = D_{T_1}(\vec{\pi}') + D_{T_2}(\vec{\pi}'') \tag{4}$$

and for all $\lambda \geq 0$

$$J_{T_1 \cup_T T_2, \lambda}(\vec{\pi}' \times \vec{\pi}'') = J_{T_1, \lambda}(\vec{\pi}') + J_{T_2, \lambda}(\vec{\pi}''). \tag{5}$$

Finally, if T_1 is the root graph of a dependency tree T and T_2 is a direct subforest of T, and again $\vec{\pi}' = (\pi'_{r(T)})$ and $\vec{\pi}''$ are policy vectors for T_1 and T_2, respectively, then

$$D_{T_1 \cup_T T_2}(\vec{\pi}' \times \vec{\pi}'') = D_{T_1}(\vec{\pi}') + (1 - \epsilon(\pi'_{r(T)}))D_{T_2}(\vec{\pi}'') \tag{6}$$

and for all $\lambda \geq 0$

$$J_{T_1 \cup_T T_2, \lambda}(\vec{\pi}' \times \vec{\pi}'') = \begin{cases} J_{T_1, \lambda}(\vec{\pi}') + (1 - \epsilon(\pi'_{r(T)}))J_{T_2, \frac{\lambda}{1-\epsilon(\pi'_{r(T)})}}(\vec{\pi}'') & \text{if } \epsilon(\pi'_{r(T)}) < 1 \\ \lambda(R_{T_1}(\vec{\pi}') + R_{T_2}(\vec{\pi}'')) & \text{if } \epsilon(\pi'_{r(T)}) = 1. \end{cases} \tag{7}$$

3. CONVEX HULL POLICY VECTORS FOR DEPENDENCY TREES

In this section, we propose a dynamic programming algorithm for computing convex hull policy vectors for dependency trees. The algorithm exploits the following proposition.

PROPOSITION 2. *Let T be a tree, $\vec{\pi}$ be a policy vector for T, T_s be a direct subforest of T, and $\vec{\pi}_s$ be the policy subvector of $\vec{\pi}$ for T_s. If $\vec{\pi}$ is a convex hull policy vector for T, then $\vec{\pi}_s$ is a convex hull policy vector for T_s.*

Proof. We prove the result by contradiction. Let T_r be the root graph of T and $\vec{\pi}_r = (\pi_{r(T)})$ be the policy subvector of $\vec{\pi}$ for T_r. Suppose that $\vec{\pi}_s$ is not a convex hull policy vector for T_s, that is, for each $\lambda' \geq 0$, there exists a policy vector $\vec{\pi}'_s$ for T_s such that $J_{T_s, \lambda'}(\vec{\pi}'_s) < J_{T_s, \lambda'}(\vec{\pi}_s)$. If T_s is not the complete subforest of T, then there exists a direct subforest T_t of T such that $V(T_s) \cap V(T_t) = \emptyset$ and $T_s \cup_T T_t$ is the complete subforest of T. Let $\vec{\pi}_t$ be the policy subvector of $\vec{\pi}$ for T_t. Then $T = T_r \cup_T (T_s \cup_T T_t)$ and $\vec{\pi} = \vec{\pi}_r \times (\vec{\pi}_s \times \vec{\pi}_t)$.

Assume that $\epsilon(\pi_{r(T)}) < 1$. Then for $\lambda \geq 0$, let $\lambda' = \frac{\lambda}{1-\epsilon(\pi_{r(T)})}$, $\vec{\pi}'_s$ be a policy vector for T_s such that $J_{T_s, \lambda'}(\vec{\pi}'_s) < J_{T_s, \lambda'}(\vec{\pi}_s)$, and $\vec{\pi}' = \vec{\pi}_r \times (\vec{\pi}'_s \times \vec{\pi}_t)$. Using (5) and (7), we have

$$\begin{aligned} J_{T, \lambda}(\vec{\pi}') &= J_{T_r, \lambda}(\vec{\pi}_r) + (1 - \epsilon(\pi_{r(T)}))(J_{T_s, \frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}'_s) + J_{T_t, \frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_t)) \\ &< J_{T_r, \lambda}(\vec{\pi}_r) + (1 - \epsilon(\pi_{r(T)}))(J_{T_s, \frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_s) + J_{T_t, \frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_t)) = J_{T, \lambda}(\vec{\pi}). \end{aligned}$$

Hence, $\vec{\pi}$ cannot be a convex hull policy vector for T.

Assume that $\epsilon(\pi_{r(T)}) = 1$. Let $\vec{\pi}'_s$ be a policy vector for T_s such that $R_{T_s}(\vec{\pi}'_s) < R_{T_s}(\vec{\pi}_s)$, and $\vec{\pi}' = \vec{\pi}_r \times (\vec{\pi}'_s \times \vec{\pi}_t)$. It follows from (5) and (7) that for all $\lambda \geq 0$,

$$J_{T,\lambda}(\vec{\pi}') = \lambda(R_{T_r}(\vec{\pi}_r) + R_{T_s}(\vec{\pi}'_s) + R_{T_t}(\vec{\pi}_t)) < \lambda(R_{T_r}(\vec{\pi}_r) + R_{T_s}(\vec{\pi}_s) + R_{T_t}(\vec{\pi}_t)) = J_{T,\lambda}(\vec{\pi}).$$

Again, $\vec{\pi}$ cannot be a convex hull policy vector for T.

If T_s is the complete subforest of T, then $T = T_r \cup_T T_s$ and $\vec{\pi} = \vec{\pi}_r \times \vec{\pi}_s$. Assume that $\epsilon(\pi_{r(T)}) < 1$. For $\lambda \geq 0$, let $\lambda' = \frac{\lambda}{1-\epsilon(\pi_{r(T)})}$, $\vec{\pi}'_s$ be a policy vector for T_s such that $J_{T_s,\lambda'}(\vec{\pi}'_s) < J_{T_s,\lambda'}(\vec{\pi}_s)$, and $\vec{\pi} = \vec{\pi}_r \times \vec{\pi}'_s$. With (7),

$$J_{T,\lambda}(\vec{\pi}') = J_{T_r,\lambda}(\vec{\pi}_r) + (1-\epsilon(\pi_{r(T)}))J_{T_s,\frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}'_s) < J_{T_r,\lambda}(\vec{\pi}_r) + (1-\epsilon(\pi_{r(T)}))J_{T_s,\frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_s) = J_{T,\lambda}(\vec{\pi}).$$

Again, $\vec{\pi}$ cannot be a convex hull policy vector for T.

Assume that $\epsilon(\pi_{r(T)}) = 1$. Let $\vec{\pi}'_s$ be a policy vector for T_s such that $R_{T_s}(\vec{\pi}'_s) < R_{T_s}(\vec{\pi}_s)$, and $\vec{\pi}' = \vec{\pi}_r \times \vec{\pi}'_s$. With (7), for all $\lambda \geq 0$,

$$J_{T,\lambda}(\vec{\pi}') = \lambda(R_{T_r}(\vec{\pi}_r) + R_{T_s}(\vec{\pi}'_s)) < \lambda(R_{T_r}(\vec{\pi}_r) + R_{T_s}(\vec{\pi}_s)) = J_{T,\lambda}(\vec{\pi}).$$

Also in this case, $\vec{\pi}$ cannot be a convex hull policy vector for T. □

Let T be a dependency tree with direct subtrees $T_1, \ldots, T_k$. Proposition 2 allows us to compute convex hull policy vectors for T as follows. In a preprocessing step, we determine the set Γ of all convex hull policies and store it in a data structure where the policies are sorted by cost. This can be done, for example, with a branch and bound method [8]. Then, exploiting (5) and (7), we can find a convex hull policy vector that minimizes $J_{T,\lambda}$ for some $\lambda \geq 0$ by recursively minimizing

$$J_{T,\lambda}(\vec{\pi}) = \begin{cases} (1-\epsilon(\pi_{r(T)}))\left(-\Delta D_{r(T)} + \sum_{i=1}^{k} J_{T_i,\frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_i)\right) + \lambda B_{r(T)}\rho(\pi_{r(T)}) & \text{if } \epsilon(\pi_{r(T)}) < 1 \\ \lambda\left(B_{r(T)}\rho(\pi_{r(T)}) + \sum_{i=1}^{k} R_{T_i}(\vec{\pi}_i)\right) & \text{if } \epsilon(\pi_{r(T)}) = 1 \end{cases} \quad (8)$$

where $\vec{\pi}_1, \ldots, \vec{\pi}_k$ are the policy subvectors of $\vec{\pi}$ for the subtrees $T_1, \ldots, T_k$, respectively. Since all policies of a convex hull policy vector are convex hull policies (Lemma 3 [8]), one minimization of (8) can be done with $|\Gamma|$ evaluations of (8), one evaluation with each policy $\pi_{r(T)}$ in Γ.

Although the above approach is feasible, it is much more efficient to compute the complete set of convex hull policy vectors at once. The idea is to use a dynamic programming scheme that computes the sets of convex hull policy vectors for all full subtrees of T, starting at the leaves. In this way, the policy vectors that minimize $J_{T_i,\frac{\lambda}{1-\epsilon(\pi_{r(T)})}}(\vec{\pi}_i)$ or $R_{T_i}(\vec{\pi}_i)$ in (8) can be found by a simple look-up operation in the set of convex hull policy vectors of T_i. Ramchandran and Vetterli [14] proposed an interval subdivision method to find the point on the convex hull that minimizes the distortion for a given rate constraint, using only minimizations of a Lagrangian for different values of λ. This method can easily be extended to find the set of all points on the convex hull. We use this method to compute the set of convex hull policy vectors for a tree T when the sets of convex hull policy vectors of all direct subtrees of T have already been found. Let $\vec{\Gamma}_T$ denote the set of convex hull policy vectors for T. Let $T_1, \ldots, T_k$ be the direct subtrees of T and $\vec{\Gamma}_{T_1}, \ldots, \vec{\Gamma}_{T_k}$ denote the sets of convex hull policy vectors for $T_1, \ldots, T_k$. If $\vec{\Gamma}_{T_1}, \ldots, \vec{\Gamma}_{T_k}$ are stored in a data structure where the policy vectors are sorted by expected rate, then $J_{T,\lambda}(\vec{\pi})$ can be minimized in $O(|\Gamma| \sum_{i=1}^{k} \log |\vec{\Gamma}_{T_i}|)$ time. The set $\vec{\Gamma}_T$ can be computed from Γ, $\vec{\Gamma}_{T_1}, \ldots, \vec{\Gamma}_{T_k}$ with the recursive procedure *SubCHBuild*$(T, \{\vec{\Gamma}_{T_1}, \ldots, \vec{\Gamma}_{T_k}\}, \vec{\pi}_l, \vec{\pi}_u)$ that finds all convex hull policy vectors for T whose rates are between $R_T(\vec{\pi}_l)$ and $R_T(\vec{\pi}_u)$. The procedure does a recursive bisection search on λ and consists of the following steps.

1. Insert $\vec{\pi}_l$ and $\vec{\pi}_u$ into $\vec{\Gamma}_T$ such that $\vec{\Gamma}_T$ is sorted by expected rate.

2. Compute $\lambda = -\frac{D_T(\vec{\pi}_l) - D_T(\vec{\pi}_u)}{R_T(\vec{\pi}_l) - R_T(\vec{\pi}_u)}$ and find $\vec{\pi}^* = \arg\min_{\vec{\pi}} J_{T,\lambda}(\vec{\pi})$ by minimizing (8).

3. If $\vec{\pi}^* \neq \vec{\pi}_l$ and $\vec{\pi}^* \neq \vec{\pi}_u$, then recursively call *SubCHBuild*$(T,\{\vec{\Gamma}_{T_1},\ldots,\vec{\Gamma}_{T_k}\},\vec{\pi}_l,\vec{\pi}^*)$ and *SubCHBuild*$(T,\{\vec{\Gamma}_{T_1},\ldots,\vec{\Gamma}_{T_k}\},\vec{\pi}^*,\vec{\pi}_u)$.

The recursion starts with *SubCHBuild*$(T,\{\vec{\Gamma}_{T_1},\ldots,\vec{\Gamma}_{T_k}\},(\mathbf{0},\ldots,\mathbf{0}),(\mathbf{1},\ldots,\mathbf{1}))$ where $\mathbf{0}$ is a convex hull policy with minimum rate and $\mathbf{1}$ is a convex hull policy with maximum rate. Finding $\vec{\Gamma}_T$ with *SubCHBuild* takes $O(|\vec{\Gamma}_T|)$ minimizations of (8). So computing $\vec{\Gamma}_T$ from Γ and $\vec{\Gamma}_{T_i}$, $i = 1,\ldots,k$ can be done in $O(|\vec{\Gamma}_T||\Gamma|\sum_{i=1}^{k}\log|\vec{\Gamma}_{T_i}|)$ time.

To determine the set $\vec{\Gamma}_T$ of convex hull policy vectors for T, we compute $\vec{\Gamma}_{T_j}$ for all full subtrees T_j of T, starting at the leaves of T. This is done with the following recursive procedure *CHBuild*(T)

1. If $r(T) \in L(T)$, then T has only one vertex, thus, set $\vec{\Gamma}_T = \{(\pi)|\pi \in \Gamma\}$.

2. If $r(T) \notin L(T)$, then

   (a) for each direct subtree T_i of T, $i = 1,\ldots,k$, set $\vec{\Gamma}_{T_i} =$ *CHBuild*(T_i),

   (b) compute $\vec{\Gamma}_T =$ *SubCHBuild*$(T,\{\vec{\Gamma}_{T_1},\ldots,\vec{\Gamma}_{T_k}\},(\mathbf{0},\ldots,\mathbf{0}),(\mathbf{1},\ldots,\mathbf{1}))$.

3. Return $\vec{\Gamma}_T$.

If we exclude the preprocessing step, the overall time complexity of this approach is $O(|V(T)||\Gamma|k\max_{T_j}(|\vec{\Gamma}_{T_j}|\log|\vec{\Gamma}_{T_j}|))$, where T_j is a full subtree of T and k is the maximum out-degree of a vertex in T. Since $|\vec{\Gamma}_{T_j}|$ is bounded by $|\Gamma|^{|V(T)|}$ and k is bounded by $|V(T)|$, the worst case time complexity is bounded by $O(|V(T)|^3|\Gamma|^{|V(T)|+1}\log|\Gamma|)$. This bound is larger than the worst case time complexity of the branch and bound algorithm of [8] for computing the set of convex hull policy vectors, which is $O(|V(T)|^2|\Gamma|^{|V(T)|})$ when applied to T. In practice, however, $|\vec{\Gamma}_{T_j}|$ is very small compared to $|\Gamma|^{|V(T)|}$, making our new approach significantly faster.

4. OPTIMAL POLICY VECTORS FOR DEPENDENCY TREES

In this section, we propose a dynamic programming algorithm for computing all optimal policy vectors for a dependency tree. The algorithm exploits the following proposition.

PROPOSITION 3. *Let T be a tree, $\vec{\pi}$ be a policy vector for T, T_s be a direct subforest of T, and $\vec{\pi}_s$ be the policy subvector of $\vec{\pi}$ for T_s. If $\vec{\pi}$ is optimal for T, then $\vec{\pi}_s$ is optimal for T_s.*

Proof. We prove the result by contradiction. Let T_r be the root graph of T, and $\vec{\pi}_r = (\pi_{r(T)})$ be the policy subvector of $\vec{\pi}$ for T_r. Suppose that $\vec{\pi}_s$ is not optimal for T_s, that is, there exists a policy vector $\vec{\pi}'_s$ for T_s such that $D_{T_s}(\vec{\pi}'_s) \leq D_{T_s}(\vec{\pi}_s)$ and $R_{T_s}(\vec{\pi}'_s) < R_{T_s}(\vec{\pi}_s)$. If T_s is not the complete subforest of T, then there exists a direct subforest T_t of T such that $V(T_s) \cap V(T_t) = \emptyset$ and $T_s \cup T_t$ is the complete subforest of T. Let $\vec{\pi}_t$ be the policy subvector of $\vec{\pi}$ for T_t. Then $T = T_r \cup_T (T_s \cup_T T_t)$ and $\vec{\pi} = \vec{\pi}_r \times (\vec{\pi}_s \times \vec{\pi}_t)$. Let now $\vec{\pi}' = \vec{\pi}_r \times (\vec{\pi}'_s \times \vec{\pi}_t)$. Then, with (3),

$$R_T(\vec{\pi}') = R_{T_r}(\vec{\pi_r}) + R_{T_s}(\vec{\pi}'_s) + R_{T_t}(\vec{\pi}_t) < R_{T_r}(\vec{\pi_r}) + R_{T_s}(\vec{\pi}_s) + R_{T_t}(\vec{\pi}_t) = R_T(\vec{\pi})$$

and, with (4) and (6),

$$D_T(\vec{\pi}') = (1-\epsilon(\pi_{r(T)}))(-\Delta D_{r(T)} + D_{T_s}(\vec{\pi}'_s) + D_{T_t}(\vec{\pi}_t)) \leq (1-\epsilon(\pi_{r(T)}))(-\Delta D_{r(T)} + D_{T_s}(\vec{\pi}_s) + D_{T_t}(\vec{\pi}_t)) = D_T(\vec{\pi}).$$

So $\vec{\pi}$ cannot be optimal for T.

If T_s is the complete subforest of T, then $T = T_r \cup_T T_s$ and $\vec{\pi} = \vec{\pi}_r \times \vec{\pi}_s$. Let now $\vec{\pi}' = \vec{\pi}_r \times \vec{\pi}'_s$. Then, with (3),

$$R_T(\vec{\pi}') = R_{T_r}(\vec{\pi_r}) + R_{T_s}(\vec{\pi}'_s) < R_{T_r}(\vec{\pi_r}) + R_{T_s}(\vec{\pi}_s) = R_T(\vec{\pi})$$

and, with (6),

$$D_T(\vec{\pi}') = (1-\epsilon(\pi_{r(T)}))(-\Delta D_{r(T)} + D_{T_s}(\vec{\pi}'_s)) \leq (1-\epsilon(\pi_{r(T)}))(-\Delta D_{r(T)} + D_{T_s}(\vec{\pi}_s)) = D_T(\vec{\pi}).$$

Again, $\vec{\pi}$ cannot be optimal for T. □

Proposition 3 allows us to find the set $\vec{\Pi}_T$ of all policy vectors that are optimal for a dependency tree T as follows. In a preprocessing step, we compute the set Π of all optimal policies and store it in a data structure where the policies are sorted by cost. This can be done, for example, with a branch and bound method [8]. Now let $T_1, \ldots, T_k$ be subtrees of T such that $V(T_i) \cap V(T_j) = \emptyset$ for $i \neq j$, $i, j \in \{1, \ldots, k\}$ and let $\vec{\Pi}_{T_i}$ be the set of optimal policy vectors for T_i. We define the product of $\vec{\Pi}_{T_i}$ and $\vec{\Pi}_{T_j}$ as $\vec{\Pi}_{T_i} \times \vec{\Pi}_{T_j} = \{\vec{\pi}_1 \times \vec{\pi}_2 | \vec{\pi}_1 \in \vec{\Pi}_{T_i}, \vec{\pi}_2 \in \vec{\Pi}_{T_j}\}$ and denote by $\vec{\Pi}_{T_i} \otimes_T \vec{\Pi}_{T_j}$ the subset of policy vectors of $\vec{\Pi}_{T_i} \times \vec{\Pi}_{T_j}$ that are optimal for $T_i \cup_T T_j$. If the policy vectors of $\vec{\Pi}_{T_i}$ and $\vec{\Pi}_{T_j}$ are sorted by expected cost, then $\vec{\Pi}_{T_i} \otimes_T \vec{\Pi}_{T_j}$ can be computed in $O(|\vec{\Pi}_{T_i}||\vec{\Pi}_{T_j}|)$ time.

To compute $\vec{\Pi}_T$, we use a recursive procedure *OPBuild*(T). The procedure consists of the following steps.

1. Set $\vec{\Pi}_r$ to the set of policy vectors that are optimal for the root graph of T, $\vec{\Pi}_r = \{(\pi) | \pi \in \Pi\}$.
2. If $r(T) \in L(T)$, then T has only one vertex, thus, set $\vec{\Pi}_T = \vec{\Pi}_r$.
3. If $r(T) \notin L(T)$, then
   (a) for each direct subtree $T_1, \ldots, T_k$ of T, set $\vec{\Pi}_{T_i}$ to the set of policy vectors that are optimal for T_i, $\vec{\Pi}_{T_i} =$ *OPBuild*(T_i),
   (b) set $\vec{\Pi}_{\text{csf}}$ to the set of policy vectors that are optimal for the complete subforest of T, $\vec{\Pi}_{\text{csf}} = \vec{\Pi}_{T_1} \otimes_T \vec{\Pi}_{T_2} \otimes_T \ldots \otimes_T \vec{\Pi}_{T_k}$,
   (c) set $\vec{\Pi}_T = \vec{\Pi}_r \otimes_T \vec{\Pi}_{\text{csf}}$.
4. Return $\vec{\Pi}_T$.

If we exclude the preprocessing step, the time complexity for the computation of $\vec{\Pi}_T$ with *OPBuild*(T) is $O(|V(T)| \max_{T_{\text{sf}}}(|\vec{\Pi}_{T_{\text{sf}}}|)^2)$, where T_{sf} is a subforest of T. The time complexity of *OPBuild*(T) is also bounded by $O(|V(T)||\Pi|^{|V(T)|})$, which is lower than the worst-case bound $O(|V(T)|^2|\Pi|^{|V(T)|})$ for the time complexity of the branch and bound algorithm of [8].

5. REDUCTION OF TIME AND SPACE COMPLEXITY

Both time and space complexity of the dynamic programming methods presented in the last two sections depend on the size of the intermediate results $\vec{\Gamma}_{T_i}$ (resp. $\vec{\Pi}_{T_i}$), the sets of convex hull (resp. optimal) policy vectors for the full subtrees T_i of a dependency tree T. The sizes of these sets depend on all input parameters, the dependency tree, the values of B_i and ΔD_i, $i = 1, \ldots, L$, the set of convex hull policies (resp. the set of optimal policies), but cannot be predicted from them easily. In worst-case situations, the intermediate results can grow exponentially with the number of recursions. However, in real-time applications, we have constraints on running time and memory consumption and thus, we have to control the sizes of the intermediate results regardless the actual input parameters. We propose a thinning of the intermediate results to reduce their size, if needed. This thinning may lead to sub-optimal policy vectors in the final result, but as a consequence of the following proposition, the distortion increase in the final result is bounded.

PROPOSITION 4. *If $\vec{\pi}^*$ and $\vec{\pi}'$ are policy vectors for a dependency tree T, and $\vec{\pi}^*_i$ and $\vec{\pi}'_i$ are policy subvectors of $\vec{\pi}^*$ and $\vec{\pi}'$, respectively, for a full subtree T_i of T, $T_i \neq T$, then $|D_T(\vec{\pi}') - D_T(\vec{\pi}^*)| \leq |D_{T_i}(\vec{\pi}'_i) - D_{T_i}(\vec{\pi}^*_i)|$.*

Proof. If T_i is a direct subtree of T, then let T_r be the root graph of T and T_j be the direct subforest of T with $V(T_j) = (V(T) \setminus V(T_i)) \setminus \{r(T)\}$. Let $\vec{\pi}^* = \vec{\pi}_0 \times \vec{\pi}^*_i$ and $\vec{\pi}' = \vec{\pi}_0 \times \vec{\pi}'_i$ with $\vec{\pi}_0 = \vec{\pi}_r \times \vec{\pi}_j$ where $\vec{\pi}_r$ is the

N	CBB	CDP	SA
4	8649903	1608	237
6	608689645	4086	879
8	1806849855	5120	1260

(a)

N	CDP	SA
4	4182	89
6	11007	2651
8	13190	3969

(b)

Table 1. Number of checked policy vectors by CBB [8], CDP (Section 3), and SA [1] for (a) a group of 10 packets of the Foreman video sequence encoded with MPEG1 and (b) a group of 19 packets of the Foreman video sequence encoded with H.264. The variable N denotes the number of transmission opportunities.

policy subvector of $\vec{\pi}_0$ for T_r and $\vec{\pi}_j$ is the policy subvector of $\vec{\pi}_0$ for T_j. Let $\vec{\pi}_r = (\pi_{r(T)})$. Since T_i and T_j are independent in T, we can apply (4) and (6) and get

$$|D_T(\vec{\pi}') - D_T(\vec{\pi}^*)| = |(1 - \epsilon(\pi_{r(T)}))(D_{T_i}(\vec{\pi}'_i) - D_{T_i}(\vec{\pi}^*_i))| \leq |D_{T_i}(\vec{\pi}'_i) - D_{T_i}(\vec{\pi}^*_i)|,$$

since $0 \leq \epsilon(\pi_{r(T)}) \leq 1$.

If T_i is not a direct subtree of T, then there exist full subtrees $T_{i_1}, \ldots, T_{i_k}$ of T such that $T_{i_{l+1}}$ is a direct subtree of T_{i_l} for $l = 1, \ldots, k-1$, T_{i_1} is a direct subtree of T, and T_i is a direct subtree of T_{i_k}. Let $\vec{\pi}'_{i_l}$ and $\vec{\pi}^*_{i_l}$ be the policy subvectors of $\vec{\pi}'$ and $\vec{\pi}^*$, respectively, for T_{i_l}, $l = 1, \ldots, k$. With (5) we get $|D_T(\vec{\pi}') - D_T(\vec{\pi}^*)| \leq |D_{T_{i_1}}(\vec{\pi}'_{i_1}) - D_{T_{i_1}}(\vec{\pi}^*_{i_1})| \leq \ldots \leq |D_{T_{i_k}}(\vec{\pi}'_{i_k}) - D_{T_{i_k}}(\vec{\pi}^*_{i_k})| \leq |D_{T_i}(\vec{\pi}'_i) - D_{T_i}(\vec{\pi}^*_i)|$. □

We do the thinning of $\vec{\Pi}_{T_i}$ as follows. Let $\vec{\Pi}_{T_i}(k)$, $k = 0, \ldots, |\vec{\Pi}_{T_i}| - 1$ be the kth policy vector of $\vec{\Pi}_{T_i}$ in decreasing $D_{T_i}(\vec{\Pi}_{T_i}(k))$ order. For each $k = 1, \ldots, |\vec{\Pi}_{T_i}| - 1$, we compute a value δ_k which approximates the increase in the expected distortion for all rates in the interval $[R_{T_i}(\vec{\Pi}_{T_i}(k)), R_{T_i}(\vec{\Pi}_{T_i}(k+1)))$ for $k = 1, \ldots, |\vec{\Pi}_{T_i}|-2$ or in the interval $[R_{T_i}(\vec{\Pi}_{T_i}(k)), \infty)$ for $k = |\vec{\Pi}_{T_i}| - 1$ due to discarding $\vec{\Pi}_{T_i}(k)$. From (5) we can see that the effect of discarding a policy vector on the distortions of the next intermediate or of the final result is proportional to $1 - \epsilon(\pi_r(T))$. However, $\pi_r(T)$ is not known at the time of the thinning process, but $\epsilon(\pi_r(T))$ and $D_{T_i}(\vec{\pi}_i)$ are usually highly correlated. In order to compensate for the factor $1 - \epsilon(\pi_r(T)$ in (5), we approximate it by $\gamma_k = (D_{T_i}(\vec{\Pi}_{T_i}(k)) - D_{\min})/(D_{\max} - D_{\min})$ with $D_{\max} = D_{T_i}(\vec{\Pi}_{T_i}(0))$ and $D_{\min} = D_{T_i}(\vec{\Pi}_{T_i}(|\vec{\Pi}_{T_i}| - 1))$ and set $\delta_k = (D_{T_i}(\vec{\Pi}_{T_i}(k-1)) - D_{T_i}(\vec{\Pi}_{T_i}(k)))/\gamma_k$. We want to remove policy vectors from $\vec{\Pi}_{T_i}$ for which δ_k is small. So for each removal of a policy vector, we find $k^* = \arg\min_k \delta_k$. Then, if $k^* < |\vec{\Pi}_{T_i}| - 1$, we set $\delta_{k^*+1} = \delta_{k^*+1} + \delta_{k^*}$. Finally, we remove $\vec{\Pi}_{T_i}(k^*)$ from $\vec{\Pi}_{T_i}$. Note that δ_k is always associated with $\vec{\Pi}_{T_i}(k)$, so δ_{k^*} is also removed.

For the thinning of $\vec{\Gamma}_{T_i}$, we use exactly the same method. The removals are repeated until a stopping criterion is satisfied. There are several choices for this stopping criterion. One choice is to remove policy vectors as long as $\delta_{k^*} < \delta_{\max}$. In this case, the maximal increase in distortion of the final result, compared to the optimal policy vectors, is bounded by $L\delta_{\max}$ if the thinning is done for each $\vec{\Gamma}_{T_i}$ (resp. $\vec{\Pi}_{T_i}$). Another choice is to stop as soon as $|\vec{\Gamma}_{T_i}|$ (resp. $\vec{\Pi}_{T_i}$) is below a certain size limit s. Here, both the time and the space complexities become bounded by $O(|V(T)|^2|\Gamma|s \log s)$ for *CHBuild*(T) (resp. $O(|V(T)|s^2)$ for *OPBuild*(T)), where Γ is the set of convex hull policies. A third choice of the stopping criterion is a combination of the other two. Note that it is not useful to do a thinning on the final result because this would not improve time or space complexity and the result would be worse.

6. EXPERIMENTAL RESULTS

In this section, we compare the performance of the proposed dynamic programming algorithms to that of the SA algorithm of [1] and the branch and bound algorithms of [8] when the packet dependency graph G is tree reducible. We use the number of policy vectors that were checked by an algorithm as the performance measure of the computational effort. Using the number of checked policy vectors as the performance measure slightly favors SA and the branch and bound algorithms over the dynamic programming algorithms because the dynamic programming algorithms can check any policy vector in amortized constant time, whereas SA and the branch and bound algorithms require $O(|V(G)|)$ time for each check.

We report results for two tree-reducible dependency graphs. The first one consists of 10 packets obtained by encoding 10 frames (frame 13 to frame 22) of the Foreman video sequence in CIF format according to the

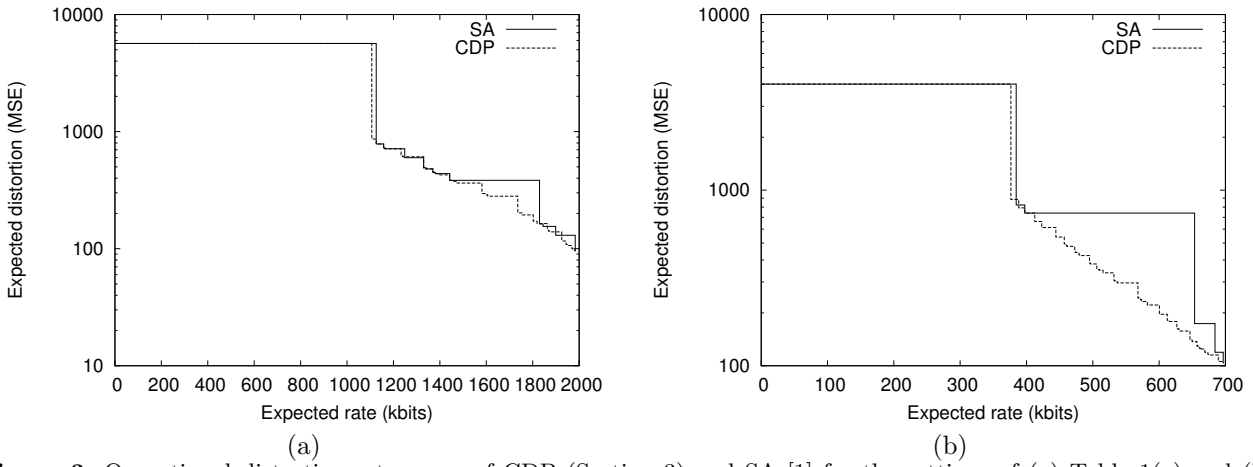


Figure 3. Operational distortion-rate curve of CDP (Section 3) and SA [1] for the settings of (a) Table 1(a) and (b) Table 1(b) with $N = 4$.

Video-CD standard (MPEG1, 1150 kilobits/s with a frame rate of 25 frames/s). Each packet corresponds to the encoding of one of the 10 frames. The frame sequence was IBBPBBPBBP, and the dependency graph is shown in Fig. 1 (a). The packet sizes (in bits) were $(B_1, \ldots, B_{10})$=(211048, 30252, 24996, 178508, 20820, 20292, 81988, 26820, 15164, 77676). The distortions were computed as follows. Let $X_i,\ i = 1, \ldots, L$, denote the block in the luminance component of the original video sequence corresponding to packet i. Let $Y_i,\ i = 1, \ldots, L$, denote the block in the luminance component of the decoded sequence corresponding to packet i. Let C denote a block of constant grey value 128. Then $D_0 = \frac{1}{L}\sum_{i=1}^{L} \mathrm{MSE}(X_i, C)$ and $\Delta D_i = \frac{1}{L}(\mathrm{MSE}(X_i, C) - \mathrm{MSE}(X_i, Y_i))$, where MSE is the mean-square-error. This gave $(D_0, \Delta D_1, \ldots, \Delta D_{10})$ =(5658.78, 560.59, 560.32, 562.07, 566.23, 567.47, 567.13, 567.41, 566.15, 566.30, 565.98). The second dependency graph consists of 19 packets obtained by encoding 19 frames (frame 13 to frame 31) of the Foreman video sequence in CIF size with H.264 at 318 kilobits/s with a frame rate of 25 frames/s. The frame sequence was IBBPBBPBBPBBPBBPBBP. The packet sizes (in bits) were $(B_1, \ldots, B_{19})$=(50152, 20424, 6024, 4160, 23704, 3208, 3096, 27056, 2584, 3080, 28688, 3624, 4904, 24424, 4256, 2288, 23680, 2824, 3360). The distortions were $(D_0, \Delta D_1, \ldots, \Delta D_{19})$ =(4018.55, 210.12, 212.19, 210.05, 210.68, 212.73, 212.88, 212.77, 212.40, 212.36, 212.48, 208.73, 210.94, 209.59, 209.14, 208.03, 208.05, 210.18, 209.88, 210.16).

A transmission policy π for a single packet was defined by a bit vector [8], and its error and cost were computed as in [8]. The time interval between two transmission opportunities was 50 ms. The network was modeled as an independent time-invariant packet erasure channel with random delays [1]. The packet loss probability of the forward and backward channels was set to 0.2. The channel forward trip time and backward trip time were modeled as shifted gamma distributed random variables FTT and BTT with rightward shifts $\kappa_F = \kappa_B = 25$ ms, and parameters $n_F = n_B = 2$, $\alpha_F = \alpha_B = \frac{1}{12.5}$ [1].

We first consider the computation of all convex hull policy vectors. Table 1 compares the performance of the SA algorithm of [1] when used to determine all convex hull policy vectors with the recursive bisection search method explained in Section 3, the branch and bound algorithm (CBB) of [8] (Appendix 7 in [13]), and the dynamic programming algorithm (CDP) of Section 3.

CDP required significantly less computational effort than CBB, but moderately more than SA. The complexity of CBB was too high to allow computation of the solutions in reasonable time when the number of packets was equal to 19. On the other hand, both CDP and CBB are exact algorithms, whereas SA is a heuristic technique, which is not guaranteed to find a convex hull policy vector (though the solution it finds might be better than a convex hull solution). Thus, the distortion-rate points reachable with SA are generally of lower quality than those reachable with CDP and CBB (Fig. 3).

We now consider the computation of all optimal policy vectors. The only previous exact method for computing all optimal policy vectors for a group of interdependent packets is a branch and bound algorithm (OBB) proposed in [8] (Appendix 9 in [13]). Table 2 compares OBB to the dynamic programming algorithm of Section 4 (ODP).

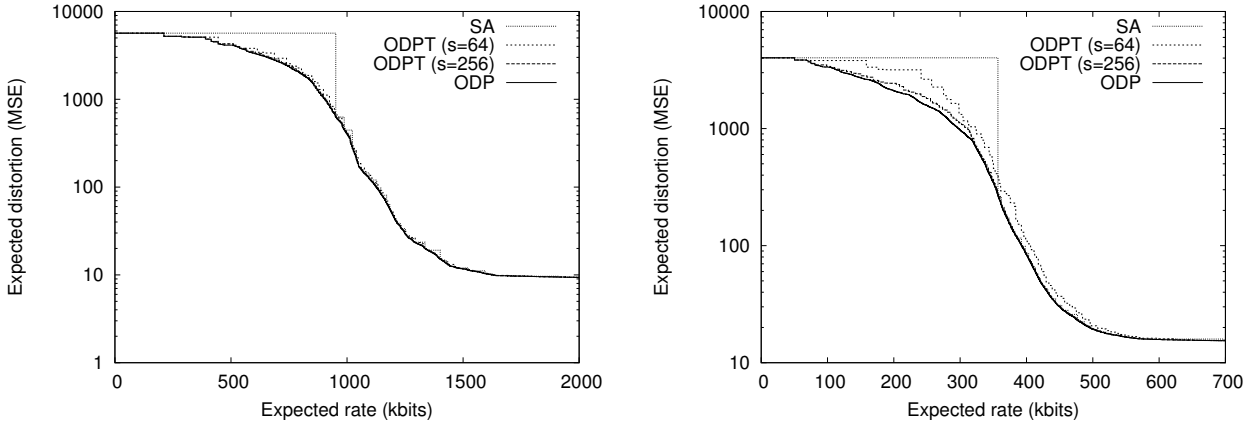
N	OBB	ODP	ODPT
4	1196627117	28071	13671
6		177606	33102
8		985896	71388

(a)

N	ODP	ODPT
4	267246	35370
6	2247552	75744
8	15697152	155484

(b)

Table 2. Number of checked policy vectors by OBB [8], the dynamic programming algorithm of Section 4 (ODP), and ODP with the thinning technique of Section 5 (ODPT) using $s = 256$ for (a) a group of 10 packets of the Foreman video sequence encoded with MPEG1 and (b) a group of 19 packets of the Foreman video sequence encoded with H.264. The variable N denotes the number of transmission opportunities.



(a) (b)

Figure 4. Operational distortion-rate curve of SA, ODPT, and ODP for the settings of Table (a) 2(a) and (b) 2(b) with $N = 8$. The thinning was done with a size limit of s for each intermediate result.

The tables also show the results of ODP when combined with the thinning technique of Section 5 (ODPT). The thinning was done with a size limit of $s = 256$ policy vectors for each intermediate result. Here also the branch and bound approach was much more complex than dynamic programming. In particular, OBB was not feasible when the size of the problem increased (with increasing number of transmission opportunities N or packets L). The thinning allowed considerable complexity reduction, which was penalized by a small quality loss of the solutions (Fig. 4).

In many situations, one is not interested in all optimal policy vectors, but only in one optimal policy vector for a given transmission rate. The previous best exact algorithm to compute one such solution is the branch and bound algorithm (BBC) of [8] (Appendix 8 in [13]). Tables 3, 4, 5, and 6 compare BBC and SA to an approach that uses dynamic programming (ODP) to compute all optimal policy vectors and then selects the best one for the given transmission rate. In SA, we used the fast convex search algorithm of Ramchandran and Vetterli [14] to find the best value of λ for a given rate. In Table 3 and 4, we do not display the rate and distortion results of BBC since this algorithm is exact and therefore finds the same (optimal) solutions as ODP. Also the complexity of ODP with and without thinning is not included because it can be read from Table 2. In many instances, BBC was too slow and we stopped it before the solution was found. The results show that ODPT, the thinning was done again with a size limit of $s = 256$ policy vectors, was able to determine high-quality approximations. The SA algorithm was always the fastest algorithm, but for low transmission rates, it found poor solutions (e.g., Table 3 with $R_{\max} = 750$ kbits or Table 6).

Finally, to give an idea on the practical speed of our dynamic programming algorithms, we report some CPU times measured on an AMD Athlon XP, 1400 MHz, PC133. To compute all convex hull policy vectors with $N = 8$ transmission opportunities, CDP needed 1.1 ms for the 10 packets of the MPEG1 encoded Foreman sequence and 3 ms for the 19 packets of the H.264 encoded Foreman sequence. To compute all optimal policy vectors with $N = 8$ transmission opportunities, ODP (resp. ODPT) needed 0.47 s (resp. 0.04 s) for the 10 packets of the MPEG1 encoded Foreman sequence and 9.27 s (resp. 0.11 s) for the 19 packets of the H.264 encoded Foreman sequence. All above times do not include a preprocessing step of 0.17 ms needed to compute Γ (resp. Π).

	SA			ODP		BBC	ODPT	
$R_{\max}$	R	D	no. checked vectors	R	D	no. checked vectors	R	D
500	0	5658.78	22	495.251	4152.53	1311	472.447	4182.65
750	0	5658.78	22	749.491	2604.65	7724229	749.491	2604.65
1000	0	5658.78	22	997.802	1391.63	55555281	997.802	1391.63
1250	1248.452	598.65	67	1248.452	598.65	106567147	1248.452	598.65
1500	1442.234	383.40	36	1496.956	348.22	59814334	1496.956	348.22
1750	1442.234	383.40	36	1749.603	197.53	4482095	1749.603	197.53
2000	1983.046	95.09	1	1983.046	95.09	19	1983.046	95.09

Table 3. Performance of SA [1], the dynamic programming algorithm of Section 4 (ODP), ODP with the thinning technique of Section 5 (ODPT) using $s = 256$, and the branch and bound algorithm (BBC) [8] for an expected transmission rate budget of $R_{\max}$ kilobits. Results are given for 10 packets of Foreman encoded with MPEG1. The number of transmission opportunities is $N = 4$. R denotes the expected rate in kilobits and D the expected MSE.

	SA			ODP		BBC	ODPT	
$R_{\max}$	R	D	no. checked vectors	R	D	no. checked vectors	R	D
500	0	5658.78	33	499.709	4129.98	42495	444.804	4273.26
750	0	5658.78	33	749.990	2358.38	168462385634	745.275	2430.23
1000	987.835	446.19	56	999.822	408.99		996.752	414.18
1250	1233.608	33.75	89	1249.856	29.62		1249.387	29.62
1500	1471.221	12.02	77	1499.630	11.69		1497.185	11.75
1750	1641.512	9.82	62	1749.278	9.66	100699134393	1744.875	9.66
2000	1908.493	9.46	49	1999.965	9.35	9499765	1991.862	9.35

Table 4. Performance of SA [1], the dynamic programming of Section 4 (ODP), ODP with the thinning technique of Section 5 (ODPT) using $s = 256$, and the branch and bound algorithm (BBC) [8] for an expected transmission rate budget of $R_{\max}$ kilobits. Results are given for 10 packets of Foreman encoded with MPEG1. The number of transmission opportunities is $N = 8$. R denotes the expected rate in kilobits and D the expected MSE.

	SA			ODP		ODPT	
$R_{\max}$	R	D	no. checked vectors	R	D	R	D
100	0	4018.55	80	98.762	3389.57	88.666	3457.53
250	0	4018.55	80	249.663	2018.47	244.838	2080.53
400	397.484	740.10	140	399.924	732.19	397.484	740.10
550	397.484	740.10	180	549.758	278.23	549.601	278.51
700	696.629	103.25	83	696.629	103.25	696.629	103.25

Table 5. Performance of SA [1], the dynamic programming algorithm of Section 4 (ODP), and ODP with the thinning technique of Section 5 (ODPT) using $s = 256$ for an expected transmission rate budget of $R_{\max}$ kilobits. Results are given for 19 packets of Foreman encoded with H.264. The number of transmission opportunities is $N = 4$. R denotes the expected rate in kilobits and D the expected MSE.

	SA			ODP		ODPT	
$R_{\max}$	R	D	no. checked vectors	R	D	R	D
100	0	4018.55	60	99.759	3348.89	98.888	3395.38
250	0	4018.55	60	249.987	1574.22	249.275	1805.67
400	398.829	86.06	24	399.992	83.78	398.756	86.36
550	546.908	16.64	24	550.000	16.57	549.508	16.63
700	697.941	15.47	1	699.239	15.40	698.019	15.41

Table 6. Performance of SA [1], the dynamic programming algorithm of Section 4 (ODP), and ODP with the thinning technique of Section 5 (ODPT) using $s = 256$ for an expected transmission rate budget of $R_{\max}$ kilobits. Results are given for 19 packets of Foreman encoded with H.264. The number of transmission opportunities is $N = 8$. R denotes the expected rate in kilobits and D the expected MSE.

7. CONCLUSION

We presented dynamic programming algorithms for computing transmission policies in media streaming systems in which the packet dependencies are modeled by trees. The proposed algorithms are exact and much faster than the branch and bound algorithms of [8], which were developed for arbitrary dependency graphs. We also suggested a simple way of reducing the complexity of the algorithms by a thinning approach. Experimental results showed that this approach allowed the computation of high-quality solutions in real-time.

Although our dynamic programming algorithms are slower than the SA algorithm [1], they can find much better solutions (optimal solutions without thinning and near-optimal solutions with thinning). We did not compare our algorithms to the technique of [7], which is faster than the SA algorithm. However, the experiments in [7] indicate that the solutions found by this technique are significantly worse than those of [1].

Not all video encoders produce tree-reducible packet dependency graphs. In the H.264 standard, for instance, more complex dependencies are allowed. We can, however, constrain the encoder to use only tree-reducible dependency graphs. Enforcing such dependencies can induce a loss in terms of rate-distortion performance, but this is compensated by the speed-up in computing the transmission policies. Also, the encoding itself may be accelerated, since the search space for motion compensation, for instance, is reduced.

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. B. Girod, M. Kalman, Y. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 549–552, Sept. 2002.
3. T. Wiegand, G. Sullivan, and A. Luthra (eds.), "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H. 264 — ISO/IEC 14496-10 AVC)", JVT-G050r1, Geneva, Switzerland, May 2003.
4. J. Chakareski, P. A. Chou, and B. Aazhang, "Computing rate-distortion optimized policies for streaming media to wireless clients", *Proc. DCC'02*, pp. 53–62, Snowbird, UT, April 2002.
5. M. Kalman, P. Ramanathan, and B. Girod, "Rate-distortion optimized video streaming with multiple deadlines", *Proc. IEEE ICIP-03*, Barcelona, Sept. 2003.
6. J. Chakareski and B. Girod, "Computing rate-distortion optimized policies for streaming media with rich acknowledgements", *Proc. DCC'04*, pp. 202–211, Snowbird, UT, April 2004.
7. J. Chakareski, J. Apostolopoulos, and B. Girod, "Low-complexity rate-distortion optimized streaming", *Proc. IEEE ICIP-2004*, Singapore, Oct. 2004.
8. M. Röder, J. Cardinal, R. Hamzaoui, "On the complexity of rate-distortion optimal streaming of packetized media", *Proc. DCC'04*, pp. 192–201, Snowbird, UT, April 2004.
9. M. Röder, J. Cardinal, and R. Hamzaoui, "Dynamic programming algorithm for rate-distortion optimized media streaming", *Proc. IEEE ICIP'05*, Genova, Italy, 2005.
10. A.V. Aho, M.R. Garey, and J.D. Ullman, "The transitive reduction of a directed graph", SIAM J. Comput., vol. 1, no. 2, June 1972.
11. A. Goralčíková and V. Koubek, "A reduct and closure algorithm for graphs", *Proc. Int. Symp. Mathematical Foundations of Computer Science*, LNCS 74, pp. 301–307, 1979.
12. A. Seghal and P. Chou, "Rate-distortion optimized streaming over diffserv networks", *Proc. IEEE ICME 02*, Lausanne, Switzerland, August 2002.
13. M. Röder, J. Cardinal, and R. Hamzaoui, "On the complexity of rate-distortion optimal streaming of packetized media", Konstanzer Schriften in Mathematik und Informatik, No. 195, Jan. 2004, `http://www.inf.uni-konstanz.de/Preprints/`.
14. K. Ramchandran and M. Vetterli, "Best wavelet packet bases in a rate-distortion sense", IEEE Transactions on Image Processing, vol. 2, no. 2, pp. 160–175, April 1993.