CONSTRUCTING DEPENDENCY TREES FOR RATE-DISTORTION OPTIMIZED MEDIA STREAMING

Martin Röder[1], *Jean Cardinal*[2], *Raouf Hamzaoui*[1]

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

ABSTRACT

Finding adequate packet transmission strategies for media streaming systems is a challenging algorithmic task. Recently, we proposed an efficient dynamic programming algorithm for streams in which the dependencies between packets, such as those prescribed between video frames by video codecs, can be modeled with a tree. In this contribution, we propose a heuristic algorithm for arbitrary dependency graphs. This algorithm consists of first transforming the dependency graph into a tree by adding dependencies, and then applying the dynamic programming algorithm on the tree thus obtained. The algorithm is both simple and efficient, as shown by experimental results on video sequences.

1. INTRODUCTION

A media streaming system allows the receiver to play back a compressed media bitstream continuously after only a short delay. The research on streaming media content over the Internet has covered various issues, including congestion control, error control, protocols, media synchronization, and source compression. This paper focuses on error control. Specifically, it deals with the problem of finding efficient transmission strategies for a pre-encoded packetized media bitstream within the framework introduced by Chou and Miao [1]. This framework models the packet dependencies with a graph, assigns an expected transmission cost and an expected reduction in distortion for each packet, and characterizes an optimal transmission strategy for the group of packets as one that minimizes the expected overall distortion for a given expected transmission cost.

In their pioneering work, Chou and Miao [1] gave a fast heuristic iterative algorithm called Sensitivity Adaptation (SA) for solving this optimization problem. Unfortunately, the solutions found by the SA algorithm can be poor [2]. Optimal solutions can be computed with a branch and bound algorithm [2]. However, the time complexity of this algorithm is too high when the number of packets is not small. Chakareski, Apostolopoulos, and Girod [3] proposed an algorithm that is faster than the SA algorithm, but its solutions are significantly worse. In [4], we showed that for packets whose dependency graph is tree reducible, optimal solutions can be computed much more efficiently than with the branch and bound algorithm of [2].

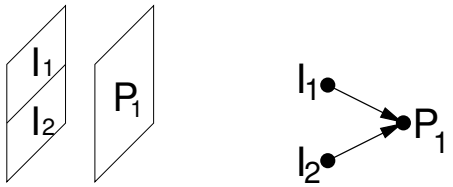

Fig. 1. (a) Slicing in H.264. (b) Dependency graph for (a).

In many situations, however, the dependency graph is not tree reducible. For example, the H.264 standard allows to partition a frame into a number of slices which can be decoded independently of each other. One reason for this partitioning is to create separately decodable packets that do not exceed a certain size limit prescribed by the network. An example is shown in Figure 1(a). The first frame is an I-frame that was partitioned into two slices I_1 and I_2 to make each slice fit into a network packet. The second frame, a P-frame, has a smaller encoded size and contains only one slice P_1. Since the P-frame is predictively encoded from the I-frame, that is, from both slices I_1 and I_2, we get the dependency graph of Figure 1(b), which is not tree reducible.

For dependency graphs that are not tree reducible, we propose an algorithm that transforms their transitive reduction into a tree. Then we apply the fast method of [4] to compute optimal solutions for the tree. Our experimental results show that these solutions are almost optimal for the original dependency graph.

2. PRELIMINARIES

The dependency between the packets that have to be transmitted 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. For $(i', i) \in E$, we say that i' is a *predecessor* of i in G. 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 and i is a *descendant* of i'. We also use the notation $i' \preceq_G i$ if $i' \prec_G i$ or $i' = i$. We denote the set of vertices of a DAG G by $V(G)$ and the set of its edges by $E(G)$. The *in-degree* of a vertex $i \in V(G)$ is the number of edges that end 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 [5]. We call an edge (i', i) of a DAG G *redundant* if there is a path from i' to i that

does not contain (i', i). Note that the transitive reduction of a DAG $G = (V, E)$ is the DAG $G = (V, E')$ where $E' = E \setminus \{(i', i) \in E \text{ and } (i', i) \text{ is redundant}\}$.

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. A DAG whose transitive reduction is a tree is said to be *tree reducible*.

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. Note that any dependency graph is equal to its transitive closure. The transitive reduction of a tree reducible dependency graph G is called the *dependency tree* of G.

Each single packet in V can be transmitted at a given number of transmission opportunities. The transmission of each 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 of the set of packets $V = \{1, \ldots, L\}$ is controlled by a policy vector $\vec{\pi} = (\pi_1, \ldots, \pi_L)$, where π_i, $i = 1, \ldots, L$, is the transmission policy for packet $i \in V$. 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. Let G be a DAG with set of vertices V. The expected cost of the transmission (the expected rate) of V with policy vector $\vec{\pi}$ is

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

and the expected distortion with respect to G is

$$D_G(\vec{\pi}) = D_0 - \sum_{i \in V} \Delta D_i \prod_{i' \preceq_G i} (1 - \epsilon(\pi_{i'})) \tag{2}$$

where D_0 is the distortion for V if no packet is received.

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 $R(\vec{\pi}) < R(\vec{\pi}^*)$. 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 $J_{G,\lambda}(\vec{\pi}) = D_G(\vec{\pi}) + \lambda R(\vec{\pi})$. All convex hull policy vectors are optimal, but not all optimal policy vectors are convex hull policy vectors.

When the dependency graph is tree reducible, we can compute the set of all optimal policy vectors using dynamic programming on subtrees: optimal policy vectors for packets in subtrees of the dependency tree are iteratively combined and pruned to form policy vectors for bigger subtrees. The complexity of this method is polynomial in the maximum number of policy vectors for a subtree. For a detailed description of the method, the reader is referred to [4].

3. PROPOSED ALGORITHM

If the dependency graph G is tree reducible, one can show [4] that its dependency tree T satisfies $D_T(\vec{\pi}) = D_G(\vec{\pi})$ for all policy vectors $\vec{\pi}$. Thus, optimal policy vectors for T are also optimal for G (the expected rate is the same for T and G). Since one can efficiently compute optimal policy vectors for trees [4], our problem is solved. However, if the transitive reduction of G is not a tree (i.e., G is not tree reducible), we usually cannot construct a tree having the same nodes as G that satisfies the above equality. Nevertheless, we will show that it is possible to construct a tree T such that a policy vector that is optimal for T does not result in a worse expected distortion when used for G. Once this tree is available, we apply the dynamic programming algorithms of [4] to compute optimal policy vectors for the tree and use these policy vectors as heuristic solutions for the original graph G. In the remaining of the section, we explain how to construct one such tree.

Szwarcfiter [6] showed that a DAG is tree reducible if and only if its transitive closure does not contain the graph shown in Figure 2(a) as an induced subgraph. We call this graph the forbidden subgraph. Let G be a dependency graph that is not tree reducible. To eliminate all forbidden subgraphs from G, we have to decide for each pair of independent vertices $\{a, b\} \subset V(G)$ having a common descendant whether we add (a, b) or (b, a) to $E(G)$. The choices should be compatible with G, thus not creating any cycle. A natural way of choosing the edges to add is using the stream order since it usually is compatible with the dependency graph and it usually places more important packets before less important ones. If the stream order is not compatible with the dependency graph, we can use any other compatible total order, such as one obtained from a topological sorting of the dependency graph. Let S be the chosen compatible order and let $S(a)$ denote the rank of packet a in S. If $S(a) < S(b)$, we add (a, b). Otherwise, we add (b, a). When G is connected, the transitive reduction of the DAG resulting from these edge additions is a tree and unique by definition. We denote this tree by $T(G, S)$.

Instead of eliminating the forbidden subgraphs from the original dependency graph G, it is more convenient to eliminate them from the transitive reduction of G. In this way, we will not add any redundant edges, and we will directly get the tree that we are interested in. This approach is justified by the fact that a DAG is tree reducible if and only if its transitive reduction does not contain the forbidden subgraph. Indeed, if the transitive closure of a DAG G contains the forbidden subgraph as an induced subgraph, then the transitive reduction of G must contain exactly one path p_1 from a to x and exactly one path p_2 from b to x, but no path from a to b or from b to a. Since p_1 and p_2 have the same end and different beginnings, the subgraph that is given by the union of p_1 and p_2 and hence the transitive reduction of G must contain the forbidden subgraph. Conversely, if the transitive reduction of a DAG contains the forbidden subgraph, it is clearly not a tree since the in-degree of x is greater than 1.

After computing the transitive reduction of G, we add an edge in a forbidden subgraph and transform the resulting DAG into its transitive reduction by removing all redundant edges from it. Suppose that we add the edge (a, b) (resp. (b, a)) in the forbidden subgraph of Figure 2(a). Then the edge (a, x) (resp. (b, x)) will be redundant and the forbidden subgraph becomes a chain in the transi-

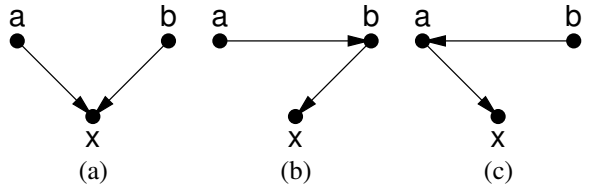

Fig. 2. (a) Forbidden subgraph. (b) Transitive reduction of the forbidden subgraph with (a, b) added. (c) Transitive reduction of the forbidden subgraph with (b, a) added.

tive reduction of the new DAG (see Figures 2(b) and 2(c)). This process is repeated until all forbidden subgraphs are eliminated, at which time we will have a tree T. Since we only remove redundant edges, and all added edges are between vertices that are independent and have a common descendant, the transitive closure of T is the same as the transitive closure of $T(G, S)$, hence $T = T(G, S)$.

Our algorithm for constructing a tree for an arbitrary connected dependency graph G and a total order S compatible with G can be summarized as follows.

1. Compute the transitive reduction T of G, $T = G^t$.
2. Repeat the following steps as long as T has a vertex with an in-degree greater than 1:
   (a) Pick a vertex x of $V(T)$ that has an in-degree greater than 1 and pick two predecessors a and b of x such that $S(a) < S(b)$.
   (b) Add the edge (a, b) to T.
   (c) Remove all edges $(x, y) \neq (a, b)$ where $x \preceq a$ and $b \preceq y$ from T.

The tree T that results from this algorithm satisfies $V(T) = V(G)$ and $E(T^T) \supseteq E(G)$. Furthermore, (2) gives

$$D_G(\vec{\pi}) \leq D_0 - \sum_{i \in V(T)} \Delta D_i \prod_{i' \preceq_T i} (1 - \epsilon(\pi_{i'})) = D_T(\vec{\pi}),$$

since $\{(i', i) | i' \preceq_G i\} = E(G) \subseteq E(T^T) = \{(i', i) | i' \preceq_T i\}$ and $0 \leq \epsilon(\pi) \leq 1$ for all policies π.

The properties of T are summarized in the following proposition.

Proposition 1. *Let G be a connected dependency graph. Then there exists a tree T such that $V(T) = V(G)$ and for any policy vector $\vec{\pi}$ for G, we have $D_T(\vec{\pi}) \geq D_G(\vec{\pi})$.*

The time complexity of the algorithm is $O(|V(G)|^2 + |V(G)||E(G)|)$ since the transitive reduction of G can be computed in $O(|V(G)||E(G)|)$ time [7] and the number of edges that can be added in Step 2b as well as the number of edges that can be removed in Step 2c is bounded by $|V(G)|^2$.

4. EXPERIMENTAL RESULTS

In this section, we compare the proposed approach to the SA algorithm [1] and to the branch and bound algorithm

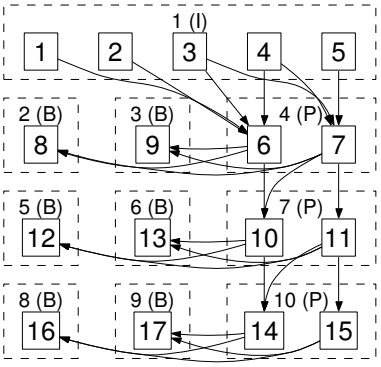

Fig. 3. Dependency graph G_1 obtained by encoding 10 frames of the Foreman video sequence with H.264. The vertex labels show the packet numbers in stream order, the dashed boxes indicate the frames with the frame number relative to the first encoded frame and the frame type in parentheses. For clarity, no redundant edges are shown.

of [2]. We used eight transmission opportunities. The time interval between two transmission opportunities was 50 ms. The delivery deadline was set to 400 ms after the first transmission opportunity for each packet. 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 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 report results for two dependency graphs that are not tree reducible. The first one, G_1 (Figure 3), consists of 17 packets obtained by encoding 10 frames (frame 13 to frame 22) 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 IBBPBBPBBP. The packet sizes (in bytes) were $(B_1, \ldots, B_{17})$=(1382, 1398, 1390, 1396, 1023, 1402, 1271, 774, 547, 1394, 664, 344, 341, 1402, 600, 232, 302). The distortions were computed as in [4], giving $(D_0, \ldots, \Delta D_{17})$ = (4042.91, 97.35, 133.11, 62.06, 71.78, 34.91, 297.09, 106.05, 399.10,400.27, 330.35, 73.56, 404.24, 404.03, 356.14, 46.99, 403.06, 403.28).

The second dependency graph G_2 is the subgraph consisting of the first nine packets of G_1, which correspond to the first four frames of the video sequence. The packet sizes $B_1, \ldots, B_9$ were the same as for G_1. The distortions were $(D_0, \ldots, \Delta D_9)$ = (4022.63, 243.37, 332.76, 155.15, 179.46, 87.28, 742.73 ,265.13 ,997.75, 1000.67).

The algorithm of Section 3 computed the tree from G_1 in 4.3 ms (Figure 4) and the tree from G_2 in 2.8 ms.

Figure 5 shows distortion-rate curves for the dependency graph G_1. The curve SA corresponds to the SA algorithm [1]. The curve ODP (resp. ODPT) corresponds to the solutions obtained by applying the dynamic programming of [4] without thinning (resp. with thinning) to the tree resulting from the algorithm of Section 3. When thinning was used, no more than 256 policy vectors were kept at each intermediate result. For rates between 0 and 23000 bytes, the SA algorithm found only the policy vector with

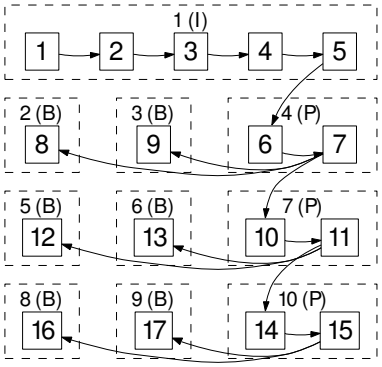

Fig. 4. Tree obtained from the dependency graph G_1.

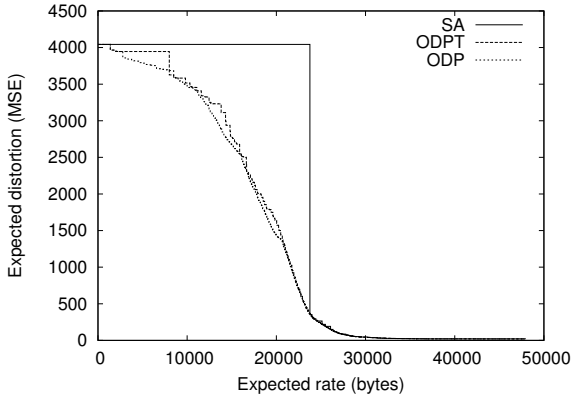

Fig. 5. Distortion-rate curves for G_1 with the SA algorithm (SA) [1] and the proposed method (without thinning in ODP, with thinning in ODPT).

expected rate 0 because it is restricted to convex-hull policy vectors. The branch and bound algorithm of [2] was not able to compute the solutions in reasonable time. The solutions found with our approach were generally better than those computed with the SA algorithm. The running times were 3.54 s for the dynamic programming algorithm without thinning, 0.08 s for the dynamic programming algorithm with thinning, and 0.02 s for the SA algorithm.

Figure 6 shows the performance of the same algorithms for the dependency graph G_2. The curve BB corresponds to the solutions obtained with the branch and bound algorithm of [2]. For most rates, the difference between the curve BB and the ODP curves was small. Since the branch and bound algorithm is exact, this indicates that our solutions were almost optimal. The running times were 3.5 days for the branch and bound algorithm, 0.16 s for the dynamic programming algorithm without thinning, 0.03 s for the dynamic programming algorithm with thinning, and 0.002 s for the SA algorithm.

5. CONCLUSION

Rate-distortion optimal policy vectors can be computed efficiently when the dependency graph of the encoded data is tree reducible [4]. We presented a fast heuristic method to compute policy vectors for dependency graphs that are not tree reducible. Our method is useful when the server cannot reencode existing packetized media data to make

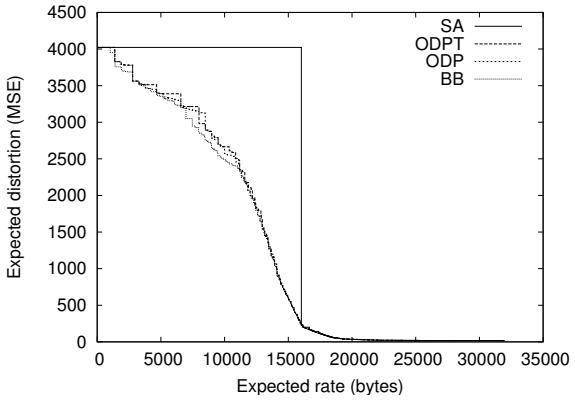

Fig. 6. Distortion-rate curves for G_2 with the SA algorithm (SA) [1], branch and bound (BB) [2], and the proposed method (without thinning in ODP, with thinning in ODPT).

their dependency graph tree reducible. This may be the case when the source encoder is not available, when changing the dependency graph leads to a drop in reconstruction quality or when time constraints are imposed. The experiments showed that our method provided high-quality solutions, which were generally better than those of the SA algorithm [1].

6. 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] M. Röder, J. Cardinal, and R. Hamzaoui, "On the complexity of rate-distortion optimal streaming of packetized media", *Proc. DCC'04*, pp. 192–201, Snowbird, UT, April 2004.

[3] J. Chakareski, J. Apostolopoulos, and B. Girod, "Low-complexity rate-distortion optimized streaming", *Proc. IEEE ICIP-2004*, Singapore, Oct. 2004.

[4] M. Röder, J. Cardinal, and R. Hamzaoui, "Efficient rate-distortion optimized media streaming for tree-reducible packet dependencies", *Proc. SPIE*, vol. 6071, MMCN'06, San Jose, Ca., Jan. 2006.

[5] 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.

[6] J.L. Szwarcfiter, "On digraphs with a rooted tree structure", *Networks*, vol. 15, pp. 49–57, 1984.

[7] 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.