Nearly Optimal Vector Quantization via Linear Programming *

(extended abstract)

Jyh-Han Lin and *Jeffrey Scott Vitter*

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

Abstract

We present new vector quantization algorithms based on the theory developed in [LiV]. The new approach is to formulate a vector quantization problem as a 0-1 integer linear program. We first solve its relaxed linear program by linear programming techniques. Then we transform the linear program solution into a provably good solution for the vector quantization problem. These methods lead to the first known polynomial-time full-search vector quantization codebook design algorithm and tree pruning algorithm with provable worst-case performance guarantees. We also introduce the notion of *pseudo-random pruned tree-structured vector quantizers*. Initial experimental results on image compression are very encouraging.

1 Introduction

A full-search vector quantizer partitions a signal space into regions each of which is represented by a representative vector [Ger, GeG, Gra]. In full-search vector quantization, the distortion between an input vector and each representative vector (codeword) in an unstructured codebook is computed. The input vector is then represented by the index of the codeword with minimum distortion. On the other hand, a tree-structured vector quantizer partitions a signal space into a hierarchy of regions. An input vector is quantized by traversing a root-to-leaf path in the tree.

*Support was provided in part by an National Science Foundation Presidential Young Investigator Award CCR–9047466 with matching funds from IBM, by NSF research grant CCR–9007851, by Army Research Office grant DAAL03–91–G–0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–91–J–4052, ARPA order 8225. The authors can be reached by electronic mail at jhl@cs.brown.edu and jsv@cs.brown.edu, respectively.

The goal of vector quantization is good data compression. The design of the codebook is a central issue in vector quantizer performance. The methods for codebook design usually involve the use of a training sequence. The training sequence is a collection of sample signal from the source to be coded. The most popular algorithm for full-search codebook design is the generalized Lloyd algorithm [GKL, LBG], an iterative clustering descent algorithm that produces a locally optimal codebook with respect to a training sequence. For tree-structured codebook, Chou, Lookabaugh, and Gray [CLG] propose a tree pruning heuristic based on the BFOS algorithm [BFO] in which a given initial tree is pruned back according to certain optimization criterion. Their heuristic traces the lower convex hull of the distortion-rate function and the final pruned subtrees are optimal for their rates. However, if there is no point (pruned subtree) on the lower convex hull at a desired rate, it requires time-sharing between two neighboring points (pruned subtrees). Lin, Storer, and Cohn [LSC] show that the tree pruning problem is $\mathcal{NP}$-hard in general.[1]

In this paper, we propose a new approach for codebook design based on linear programming. We demonstrate our methods by presenting the first known approximation algorithms with worst-case performance guarantees for a full-search codebook design problem and the tree pruning problem. An algorithm that may not lead to the optimal result is called an *approximation algorithm*. We are interested in approximation algorithms with guaranteed performance, in which we can prove that the solution they produce is not too far from the optimal solution. We remark that in practice, approximation algorithms may perform much better than their performance guarantees suggest.

Many $\mathcal{NP}$-hard optimization problems can be formulated as integer linear programs. One of the most important strategies for obtaining provably good approximation algorithms to an integer program is to drop the integrality constraints, solve the resulting linear programming problem,[2] and then round the solution to an integral solution. Much work along this line has been done, for example [Chv, Lov, Rag, RaT]. In [LiV], we build on previous work and propose new transformation methods for obtaining provably good solutions from linear program relaxation of a type of 0-1 optimization problems. These methods can be applied to codebook design problems.

In Section 2, we present a greedy full-search codebook formation algorithm. We discuss possible extensions of our algorithm for dealing with rates of codebooks and for designing *k-nearest-neighbor vector quantization* codebooks. Section 3 deals with the tree pruning problem and introduces the notion of *pseudo-random pruned tree-structured vector quantizers*. Initial experimental results on image compression are reported in Section 4. Section 5 concludes with further discussions.

[1] On the other hand, Lin, Storer, and Cohn also show that the tree pruning problem can be solved in polynomial time if the trees are binary and the cost constraint is the number of leaves. Our approximate tree pruning algorithm works for general trees and applies to other cost constraints, such as the average path length and the leaf entropy.

[2] The linear programming problem can be solved in polynomial time by the ellipsoid algorithm [Kha] or by the interior point method [Kar]. In practice, the simplex method [Dan] has been proven to be very efficient, although its worst case performance is not polynomial.

2 Full-Search Vector Quantization

In this section, we present an approximation algorithm for the *(discrete) full-search (vector quantization) codebook design problem.* We denote the signal space by (X, d_X), where d_X is a metric on X. Let $S = \{t_1, \ldots, t_n\}$ be a set of training signal data (training vectors) and let s be a given bound on the size of codebooks. The goal is to select a subset $\mathcal{U} \in S$ of s training vectors as codewords such that the mean squared error $\frac{1}{n}\sum_{i=1}^{n} \min_{t_j \in \mathcal{U}} d_X^2(t_i, t_j)$ is minimized. We remark that our algorithm also works for other distortion measures. The full-search codebook design problem can be formulated as an integer program of minimizing

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} d_X^2(t_i, t_j)x_{ij} \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{2}$$

$$\sum_{j=1}^{n} y_j \leq s, \tag{3}$$

$$x_{ij} \leq y_j, \qquad i, j = 1, \ldots, n, \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \tag{5}$$

where $y_j = 1$ if and only if t_j is chosen as a codeword, and $x_{ij} = 1$ if and only if $y_j = 1$ and t_i is quantized as codeword t_j.

The linear program relaxation of the above program is to allow y_j and x_{ij} to take rational values between 0 and 1. Clearly, the optimal fractional solution (linear program solution) is a lower bound on the solutions of the full-search codebook design problem.

The $\mathcal{NP}$-hardness result in [Pap] can be easily modified to show that the full-search codebook design problem is $\mathcal{NP}$-hard.

2.1 A Greedy Codebook Formation Algorithm

The following is the greedy codebook formation algorithm:

1. Solve the linear program relaxation of the full-search codebook design problem by linear programming techniques; denote the fractional solution by $\widehat{y}, \widehat{x}$.

2. For each i, compute $\widehat{D}_i = \sum_{j=1}^{n} d_X^2(t_i, t_j)\widehat{x}_{ij}$. Given $\epsilon > 0$, for each j such that $\widehat{y}_j > 0$, construct a set S_j. A vector t_i is in S_j if and only if $d_X^2(t_i, t_j) \leq (1+\epsilon)\widehat{D}_i$.

3. Apply the greedy set covering algorithm [Chv, Joh, Lov]: Choose the set which covers the most uncovered vectors. Repeat this process until all vectors are covered. Let U be the set of indices of sets chosen by the greedy heuristic. Output $\mathcal{U} = \{t_j\}_{j \in U}$ as the codebook.[3]

[3]The codebook can be further improved by the generalized Lloyd algorithm.

By the results in [LiV], we have the following application:

Corollary 1 *Given any $\epsilon > 0$, the greedy codebook formation algorithm outputs a codebook $\mathcal{U}$ of size at most $(1 + 1/\epsilon)s(\ln n + 1)$ such that $\frac{1}{n}\sum_{i=1}^{n} \min_{t_j \in \mathcal{U}} d_X^2(t_i, t_j) \leq (1+\epsilon)\widehat{D} \leq (1+\epsilon)D$, where $\widehat{D}$ is the mean squared error of the optimal fractional solution for the full-search codebook design problem and D is the optimal mean squared error of codebooks of size at most s.*

2.2 Extensions

2.2.1 Dealing with Rates

If the bound is on the rate of the codebook rather than on the codebook size, we may apply entropy-coding methods to the codebook produced by the greedy codebook formation algorithm. Alternatively, given a bound R on the rate of codebooks, we may formulate the codebook formation problem as a nonlinear program of minimizing

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} d_X^2(t_i, t_j)x_{ij} \tag{6}$$

subject to

$$\begin{aligned}
\sum_{j=1}^{n} x_{ij} &= 1, & i = 1, \ldots, n, & \quad (7)\\
\frac{1}{n}\sum_{i=1}^{n} x_{ij} &= p_j, & j = 1, \ldots, n, & \quad (8)\\
\sum_{j=1}^{n} p_j \log \frac{1}{p_j} &\leq R, & & \quad (9)\\
x_{ij} &\leq y_j, & i, j = 1, \ldots, n, & \quad (10)\\
x_{ij}, y_j &\in \{0, 1\}, & i, j = 1, \ldots, n, & \quad (11)
\end{aligned}$$

where $y_j = 1$ if and only if t_j is chosen as a codeword, $x_{ij} = 1$ if and only if $y_j = 1$ and t_i is quantized as codeword t_j, and p_j is the relative frequency of training vectors quantized as t_j. In this case, heuristics may be needed to solve the nonlinear program.

2.2.2 k-Nearest-Neighbor Vector Quantization

In traditional vector quantization, an input vector is quantized as its nearest codeword in the codebook. In the *k-nearest-neighbor vector quantization*, an input vector is quantized as the weighted average of its k-nearest codewords in the codebook. In the context of image compression, this mapping may have the effect of smoothing images and preventing distracting blockiness.

Given a sequence of training vectors $t_1, t_2, \ldots, t_n$ and a bound s on the codebook size, the *k-nearest-neighbor (vector quantization) codebook design problem* is to select

a subset $\mathcal{U}$ of s training vectors as codewords such that the mean squared error minimized. Our greedy codebook formation algorithm can be adapted for solving the k-nearest-neighbor codebook design problem with similar performance guarantees.

3 Tree-Structured Vector Quantization

The computational advantage of tree-structured vector quantizers (TSVQ) over full-search quantizers is that the mapping from a vector to a quantization bin can be done quickly by tree traversal. Tree-structured vector quantizers also have a distinguished "successive approximation" and "graceful degradation" character.

In this section we present an approximate tree pruning algorithm. Besides pruned tree-structured vector quantization (PTSVQ), the tree pruning problem has many other applications such as regression trees, decision trees, and computer graphics [BFO, CLG]. Our notations in this section follow that of [CLG].

A tree T is a finite set of nodes, $t_0, t_1, \ldots, t_n$, with a unique root node t_0. The set of leaves of a tree T is denoted by $\widetilde{T}$. A subtree S of tree T is a tree rooted at some node $root(S) \in T$ and the following condition holds: For each internal node t of T, if any of the children of t is in S, then all of children of t must be in S as well. The leaves $\widetilde{S}$ of a subtree S are not necessarily a subset of $\widetilde{T}$; the leaves of S may be the internal nodes of T. If $\widetilde{S} \subseteq \widetilde{T}$, then S is called a branch of T and is denoted by $T_{root(S)}$. For $t \neq t_0$, we denote the parent node of t as $parent(t)$. For $t \in T - \widetilde{T}$, let $children(t)$ be the set of children for node t. We define $path(t)$ as the set of nodes, including t, from t_0 leading to t. We call a subtree S of T a *pruned subtree* and write $S \preceq T$ if the root of S is t_0.

Definition 1 Let $u(t) \geq 0$ be an arbitrary function on the nodes of T. A *linear tree functional* u on subtrees is:

$$u(S) = \sum_{\tilde{t} \in \widetilde{S}} u(\tilde{t}).$$

Let $\Delta u(S) = u(S) - u(root(S))$. A tree functional u is monotonic nondecreasing if and only if for any subtree S of T, we have $\Delta u(S) \geq 0$. Similarly, u is monotonic nonincreasing if and only if $\Delta u(S) \leq 0$ for any subtree S of T.

Let $\mathcal{C}$ be a monotonic nondecreasing tree functional and $\mathcal{D}$ be a monotonic nonincreasing tree functional. We call $\mathcal{C}$ the *cost functional* and $\mathcal{D}$ the *distortion functional*. For example, in vector quantization, we have the following setting: Let P be a probability function such that $P(t_0) = 1$ and for all $t \in T - \widetilde{T}$, we have $P(t) = \sum_{t' \in children(t)} P(t')$. Let d be a distortion function on nodes satisfying $P(t)d(t) \geq \sum_{t' \in children(t)} P(t')d(t')$. Usually we let $\mathcal{D}(S)$ be the average distortion of subtrees. Possible definitions for $\mathcal{C}(S)$ include the average path length, the leaf entropy, or the number of leaves.

Given a tree T and a bound C on the cost, the *tree pruning problem* is to find a pruned subtree S of T such that $\mathcal{C}(S) \leq C$ and $\mathcal{D}(S)$ is minimized. We may formulate the tree pruning problem as an integer linear program as follows: For each

node $t \in T$, let x_t be a decision variable such that $x_t = 1$ if and only if node t is a leaf in the final pruned subtree, $x_t = 0$ otherwise. The integer linear program for the optimal tree pruning problem is to minimize the cost

$$\sum_{t \in T} x_t \mathcal{D}(t) \tag{12}$$

subject to

$$\sum_{t \in path(\widetilde{t})} x_t = 1, \qquad \widetilde{t} \in \widetilde{T}, \tag{13}$$

$$\sum_{t \in T} x_t \mathcal{C}(t) \leq C, \tag{14}$$

$$x_t \in \{0,1\}, \qquad t \in T. \tag{15}$$

Lin, Storer, and Cohn [LSC] show that, in general, the tree pruning problem is $\mathcal{NP}$-hard. Therefore, we have to use heuristics in practice [BFO, CLG].

3.1 Approximate Tree Pruning

The following is an outline of the approximate tree pruning algorithm:

1. Solve the linear program relaxation of the tree pruning problem by linear programming techniques; denote the fractional solution by $\widehat{x}$.
2. Given $\epsilon > 0$, in a top-down and breadth-first fashion, we prune the tree at any node t where $\sum_{t' \in path(t)} \widehat{x}_{t'} \geq 1/(1+\epsilon)$.

The results in [LiV] imply the following:

Corollary 2 *Given any $\epsilon > 0$, the approximate tree pruning algorithm outputs a pruned subtree S satisfying $\mathcal{C}(S) \leq (1 + 1/\epsilon)C$ and $\mathcal{D}(S) \leq (1+\epsilon)\widehat{D} \leq (1+\epsilon)D$, where $\widehat{D}$ is the distortion of the optimal fractional solution for the tree pruning problem and D is the optimal distortion of pruned subtrees with cost at most C.*

3.2 Pseudo-Random PTSVQ

In this section we introduce a new kind of vector quantizers called *pseudo-random pruned tree-structured vector quantizers (pseudo-random PTSVQ)*. We first define a more general notion of *probability search trees*. Probability search trees are an interesting interpretation of the fractional solution of the integer linear program for the tree pruning problem.

Definition 2 A *probability search tree* $\widehat{T} = (T, q)$ is a tree T with augmented probability function q on tree nodes, which satisfies $\sum_{t \in path(\widetilde{t})} q(t) = 1$, for all $\widetilde{t} \in \widetilde{T}$. Let us define $Q(t) = 1 - \sum_{t' \in path(t)} q(t')$. A search along a path through node t will continue at node t, assuming the search reaches node t, with probability $Q(t)/Q(parent(t))$. We may see $Q(t)$ as the probability that the search passes through node t and $q(t)$ as the probability that the search stops at node t.

We can extend tree functionals to probability search trees in the following way:

Definition 3 Let $\widehat{T} = (T, q)$ be a probability search tree. Given a linear tree functional u, we define the *probability tree functional* u^* on subtrees as

$$u^*(S) = \sum_{t \in S - \widetilde{S}} q(t)u(t) + \sum_{\widetilde{t} \in \widetilde{S}} Q(parent(\widetilde{t}))u(\widetilde{t}),$$

and we define $\Delta u^*(S) = u^*(S) - u^*(root(S))$. A probability tree functional u^* is monotonic nondecreasing if and only if for any subtree S of T, we have $\Delta u(S) \geq 0$. Similarly, u^* is monotonic nonincreasing if and only if $\Delta u(S) \leq 0$ for any subtree S of T.

The results in [LiV] imply the following monotonic properties of probability tree functionals:

Corollary 3 *If a linear tree functional u is monotonic nondecreasing (nonincreasing), then the probability tree functional u^* is also monotonic nondecreasing (nonincreasing).*

We can interpret an optimal fractional solution $\widehat{x}$ as an augmented probability function q by setting $q(t) = \widehat{x}_t$. The resulting probability search tree can be used as a pseudo-random PTSVQ, which statistically has the potential of outperforming the optimal PTSVQ in terms of the average path length:

Corollary 4 *Let $\widehat{T} = (T, q)$ be a probability search tree with $q(t) = \widehat{x}_t$, where $\widehat{x}$ is an optimal fractional solution for the tree pruning problem, and let $\mathcal{C}(S)$ be the average path length of subtrees. Then we have $\mathcal{C}^*(\widehat{T}) \leq C$ and $\mathcal{D}^*(\widehat{T}) = \widehat{D} \leq D$, where $\widehat{D}$ is the distortion of the optimal fractional solution for the tree pruning problem and D is the optimal distortion of pruned subtrees with cost at most C.*

In pseudo-random PTSVQ, the encoder and decoder use the same pseudo-random number generator for making stopping decisions. For encoding, the encoder selects a random seed r for the pseudo-random number generator and then encodes each input vector as a path according to the search procedure for probability search trees. The random seed r is transmitted along with the binary sequence. For decoding, the decoder uses the same random seed r and traverses the tree according to the encoded binary sequence and the search procedure for probability search trees.

4 Experimental Results

4.1 Full-Search Vector Quantization

The solution of linear programs dominates the time and memory requirement of the greedy codebook formation algorithm. The number of variables in the linear program is $O(n^2)$ and the number of constraints is also $O(n^2)$. A straightforward

implementation of the simplex method for linear programming requires $O(n^4)$ space. By the decomposition technique in [GNR], the space requirement can be reduced to $O(n^2)$. Unfortunately, for image compression, the number of training vectors can be in the order of 10^6. Therefore, it may require gigabytes of memory to solve the linear program. Currently we are studying ways for reducing the space requirement and for speeding up the linear programming.

Initial experimental results with hundreds of training vectors show that the greedy algorithm terminates with optimal solutions regularly. That is, with $\epsilon = 1$, often the codebook size is exactly s and the mean squared error is exactly $\widehat{D}$.

4.2 Tree-Structured Vector Quantization

The number of variables in the linear program for the tree pruning problem is $O(n)$ and the number of constraints is also $O(n)$. The space requirement is therefore $O(n^2)$. In fact, the number of nonzero entries in the constraint matrix is only $O(n \log n)$, and the space requirement can be greatly reduced by sparse-matrix techniques.

We used the USC database for our experiments. The test image was the well-known Lenna image. The code vectors were 4×4 pixel blocks. The average rate is the average path length of trees. A complete TSVQ of length 12 was designed using the training sequence. PTSVQs of average rate $0, 1, \ldots, 12$ were obtained by the approximate tree pruning algorithm. (We solved the linear programs by the Stanford MINOS package.) We also constructed a series of pseudo-random PTSVQs from the fractional solutions for the linear programs. The resulting peak signal-to-noise ratio (PSNR), which is defined as

$$10 \log_{10} \frac{(\text{peak input amplitude})^2}{\text{MSE}},$$

for each vector quantizer is plotted against its rate in Figure 1.

The results indicate that the performances of PTSVQs and pseudo-random PTSVQs are very similar. Compared with similar experiments by Riskin in [Ris], our approximate tree pruning algorithm performs at least as well as the generalized BFOS algorithm, although we use a different initial tree. We also note that the PSNRs can be further improved by predictive coding techniques as indicated in [Ris].

5 Conclusions

In this paper, we propose a new approach for vector quantization codebook design problems. Our method is to formulate a codebook design problem as a 0-1 integer linear program. We first solve its linear program relaxation and then transform the fractional solution to a provably good 0-1 solution. The codebook design problems we look into include a full-search codebook formation problem and the tree-structured vector quantizer pruning problem.

Initial experimental results indicate that our approximation algorithms perform much better in practice than their (worst-case) performance guarantees suggest. The

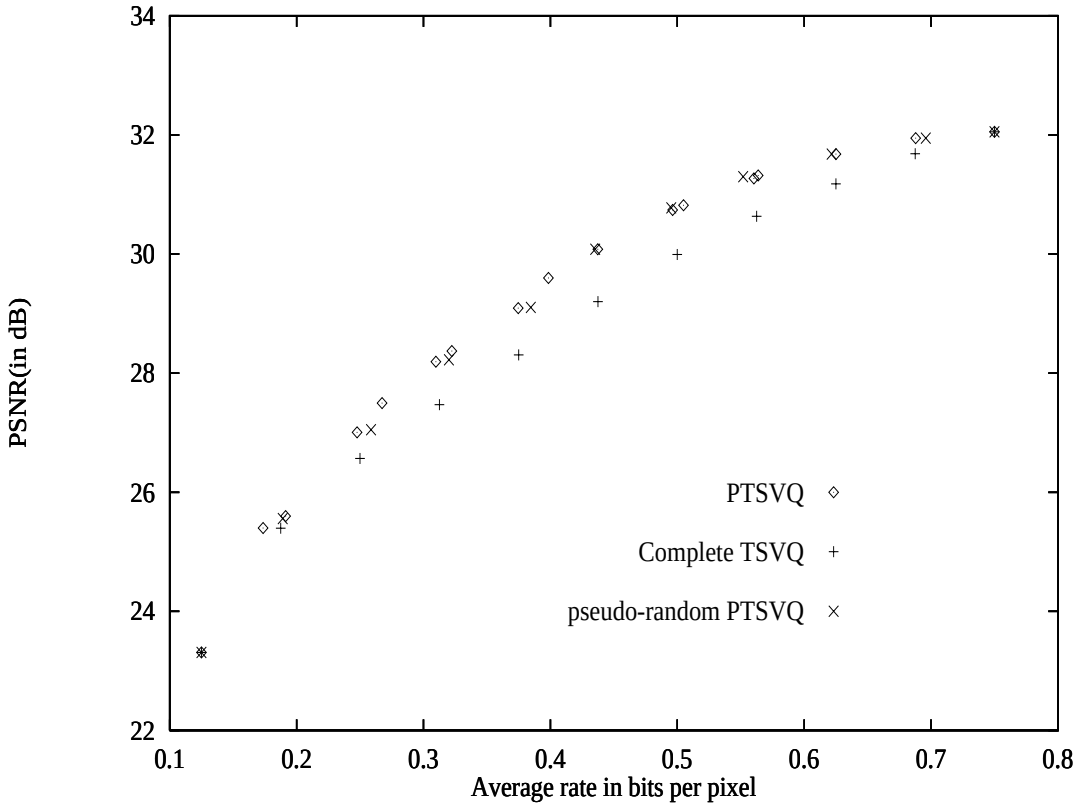

Figure 1: PSNR vs. Average Rate: USC database. The PTSVQs are obtained by the approximate tree pruning algorithm. The pseudo-random PTSVQs are constructed from the fractional solutions for the linear programs.

approximation algorithm for forming full-search codebook may not be practical at this moment due to the huge size of linear programs. On the other hand, the linear programs used by the approximate tree pruning algorithm are of moderate size and can be solved efficiently.

It is of interest to apply our approach to other problems related to vector quantization and data compression in general.

References

[BFO] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification Trees and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[CLG] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal Pruning with Applications to Tree-Structured Source Coding and Modeling," *IEEE Transactions on Information Theory* (1989), 299–315.

[Chv] V. Chvátal, "A Greedy Heuristic for the Set-Covering Problem," *Mathematics*

of Operations Research 4 (1979), 233–235.

[Dan] G. Dantzig, "Programming of Interdependent Activities, II, Mathematical Models," in *Activity Analysis of Production and Allocation*, John Wiley & Sons, Inc, New York, 1951, 19–32.

[GNR] R. S. Garfinkel, A. W. Neebe, and M. R. Rao, "An Algorithm for the M-Median Plant Location Problem," *Transportation Science* 8 (1974), 217–236.

[Ger] A. Gersho, "On the Structure of Vector Quantizers," *IEEE Transactions on Information Theory* 28 (March 1982), 157–166.

[GeG] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Press, Massachusetts, 1991.

[Gra] R. M. Gray, "Vector Quantization," *IEEE ASSP Magazine* (April 1984), 4–29.

[GKL] R. M. Gray, J. C. Kieffer, and Y. Linde, "Locally Optimal Block Quantizer Design," *Information and Control* 45 (1980), 178–198.

[Joh] D. S. Johnson, "Approximation Algorithms for Combinatorial Problems," *Journal of Computer and System Sciences* 9 (1974), 256–278.

[Kar] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," *Combinatorica* 4 (1984), 373–395.

[Kha] L. G. Khachiyan, "A Polynomial Algorithm in Linear Programming," *Soviet Math. Doklady* 20 (1979), 191–194.

[LSC] J. Lin, J. A. Storer, and M. Cohn, "On the Complexity of Optimal Tree Pruning for Source Coding," in *Proceedings of the Data Compression Conference*, J. A. Storer and J. H. Reif, eds., Snowbird, Utah, April 1991, 63–72.

[LiV] J.-H. Lin and J. S. Vitter, "ϵ-Approximations with Minimum Packing Constraint Violation," submitted for publication.

[LBG] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications* COM-28 (January 1980), 84–95.

[Lov] L. Lovász, "On the Ratio of Optimal Integral and Fractional Covers," *Discrete Mathematics* 13 (1975), 383–390.

[Pap] C. H. Papadimitriou, "Worst-case and Probabilistic Analysis of a Geometric Location Problem," *SIAM Journal on Computing* 10 (1981), 542–557.

[Rag] P. Raghavan, "Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs," *Journal of Computer and System Science* 37 (1988), 130–143.

[RaT] P. Raghavan and C. D. Thompson, "Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs," *Combinatorics* 7 (1987), 365–374.

[Ris] E. A. Riskin, *Variable Rate Vector Quantization of Images*, Ph. D. Dissertation, Stanford University, 1990.