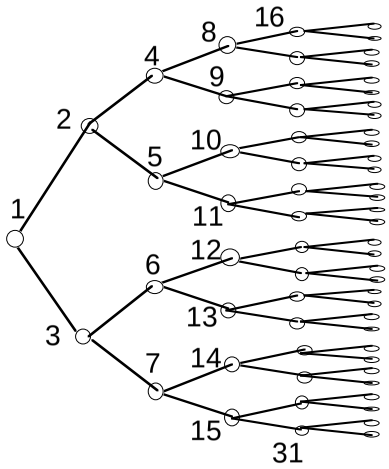
TREE-STRUCTURED VECTOR QUANTIZATION
(TSVQ)

binary search tree of depth kR

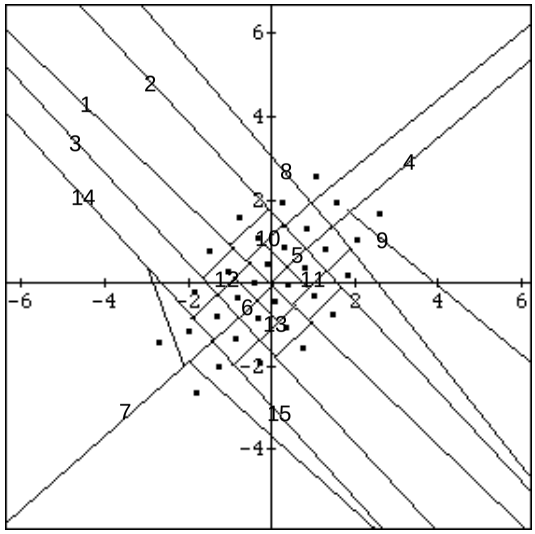


there is a hyperplane associated with each node

&

partition and codebook

k = 2, M = 32, R = 2.5



TSVQ encoding of source vector $\underline{x} = (x_1,...,x_k)$:

- Starting at the root node, perform a sequence of kR hyperplane tests.
- The hyperplane at node i: $H_i = \{ \underline{x} : \sum_{j=1}^{k} x_j h_{i,j} = t_i \}$

  where $\underline{h}_i = (h_{i,1},...,h_{i,k})$ is the "normal" to the hyperplane, and t_i is the "offset".
- Hyperplane test at node i:

  If $\sum_{j=1}^{k} x_j h_{i,j} \geq t_i$, output 0, and choose the upper child node.

  If $\sum_{j=1}^{k} x_j h_{i,j} < t_i$, output 1, and choose the lower child node.
- Move to the chosen child node.
- Stop when the chosen child node is a leaf node.
- This encoding algorithm does partitioning and binary encoding; i.e. it inputs $\underline{x}$ and outputs a kR-bit binary sequence that indexes the quant. cell containing $\underline{x}$.

TSVQ decoding:

- There is a k-dimensional codevector associated with each of the $M = 2^{kR}$ leaf nodes.
- The decoder stores the codevectors in a table, and uses the usual sort of table lookup. (The tree is not needed at the decoder.)

Notes:

- The TSVQ encoding algorithm does not usually produce a Voronoi partition, i.e. it is not optimal for the given codebook.
- Exception: Any nonuniform scalar quantizer partition can be implemented with a tree structure.

Complexity

- Arithmetic operations
  - kR hyperplane tests, each requires k multiplies, k-1 additions (and 1 comparison).
  - Arithmetic op's/sample: $\frac{2kR}{k} = 2kR = \sim 2 \log_2 M$.
  - Compare to $\sim 3M = 3 \times 2^{kR}$ for full search encoding
- Storage
  - The encoder stores the hyperplanes at each interior node: $\underline{h}_i$, t_i, $i = 1,...,M-1$.

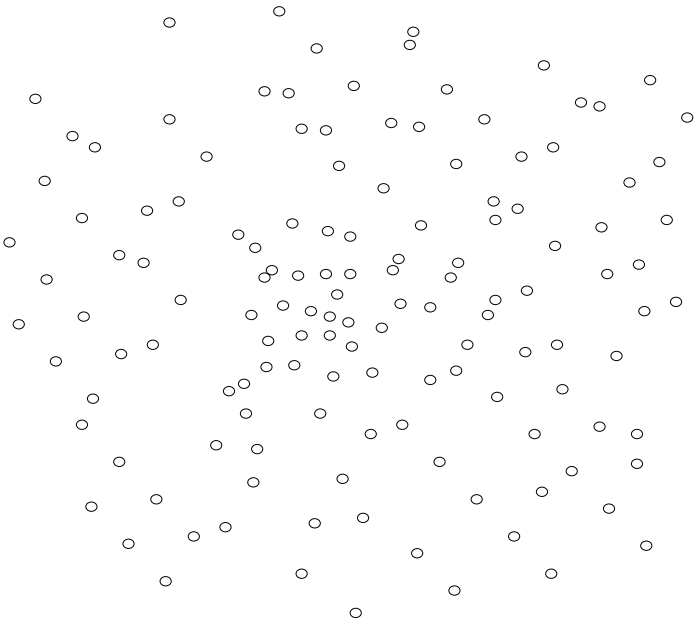
    Requires: $(M-1)(k+1)\, b \cong Mkb$ bits, where b = # bits per component

    (A tree with M leaves has M-1 interior nodes.)
  - The decoder stores the codebook of M codevectors: Mkb bits

GREEDY DESIGN ALGORITHM

Example for $k = 2$, $M = 32$

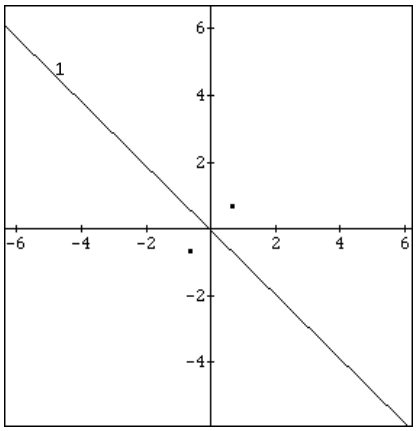
Start with a training sequence:



Choose Hyperplane 1

Apply the LBG algorithm to the training set to design the best 2 point vq.

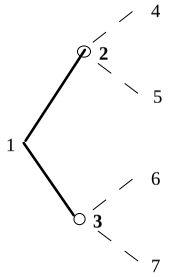
Choose hyperplane 1 to be the hyperplane that divides its cells.

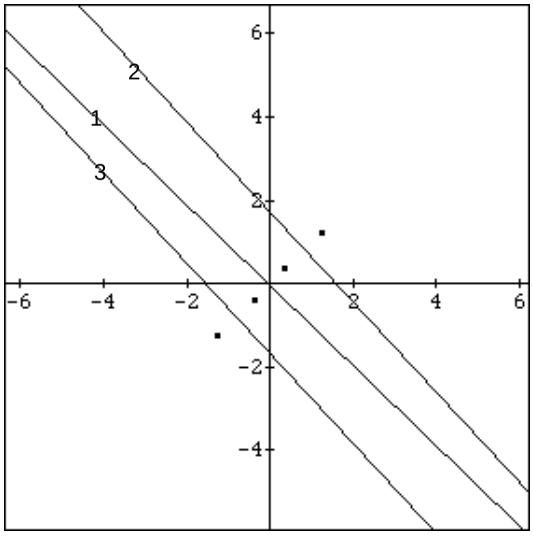


Choose Hyperplane 2

Apply the LBG algorithm to the vectors in the training set that lie above hyperplane 1 to design the best 2 point vq for these training vectors.

Choose hyperplane 2 to be the hyperplane that divides its cells.





Choose Hyperplane 3

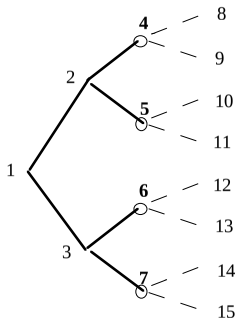
Apply the LBG algorithm to the vectors in the training set that lie below hyperplane 1 to design the best 2 point vq for these training vectors.

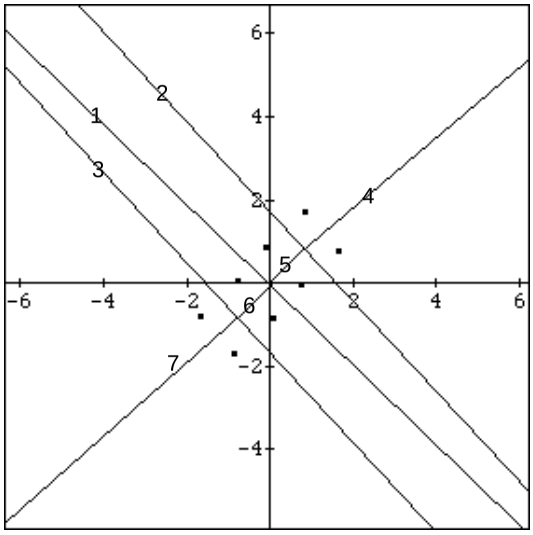
Choose hyperplane 3 to be the hyperplane that divides its cells.

Choose Hyperplane 4

Apply LBG algorithm to the vectors in training set that were prev'ly quantized to node 4, i.e. that lie above hyper-planes 1 and 2, to design the best 2 point vq for these trainingg vectors.

Choose hyperplane 4 to be the hyperplane that divides its cells.



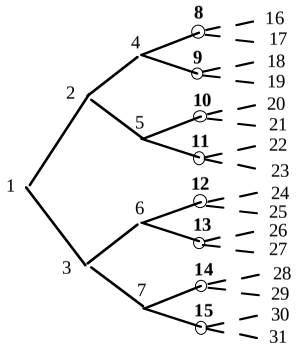


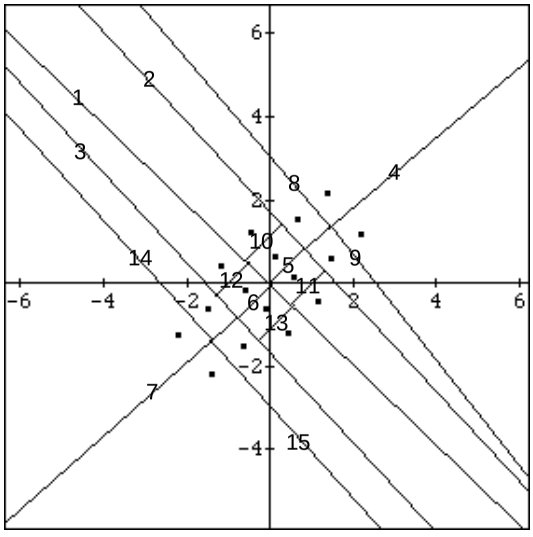
Choose Hyperplane n for n = 5,6,7

Apply the LBG algorithm to the vectors in the training set that were previously quantized to node n to design the best 2 point vq for these training vectors.

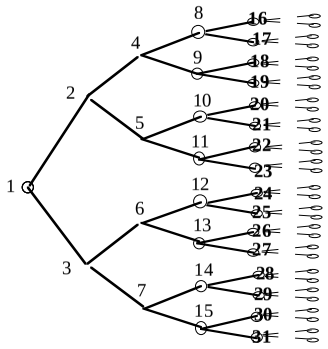
Choose hyperplane n to be the hyperplane that divides its cells.

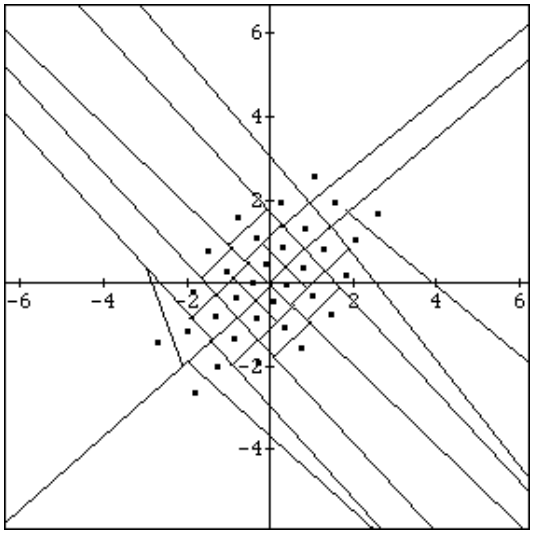
Choose Hyperplanes 8, 9, ..., 15:





Choose Hyperplanes 16, 17, ..., 31



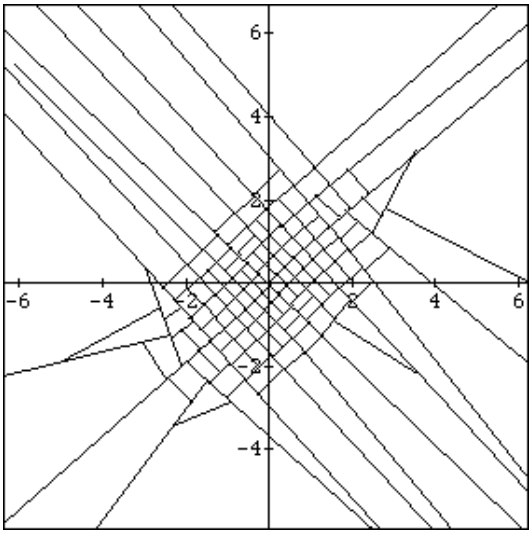


Choose codevectors $\underline{w}_1,\ldots,\underline{w}_{32}$ to be the centroids of the 32 cells associated with the 32 leaf nodes of tree. These are found in the normal course of running the LBG algorithm on nodes 16 to 31.

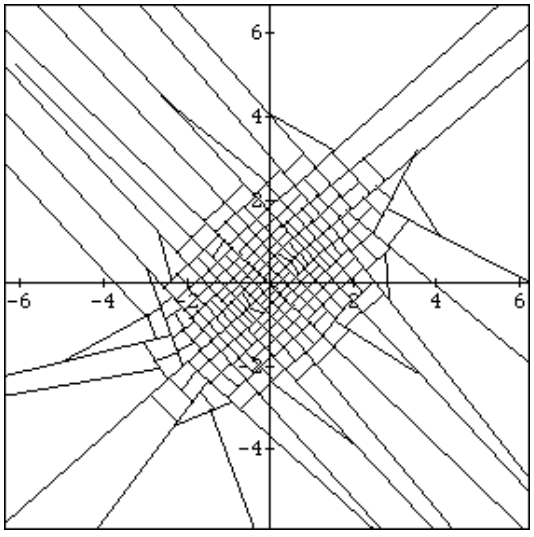
EXAMPLES OF TSVQ DESIGNED WITH THE GREEDY ALGORITHM

Gaussian AR Source, $\rho = .9$

k = 2, M = 128, R = 3.5



k = 2, M = 256, R = 4

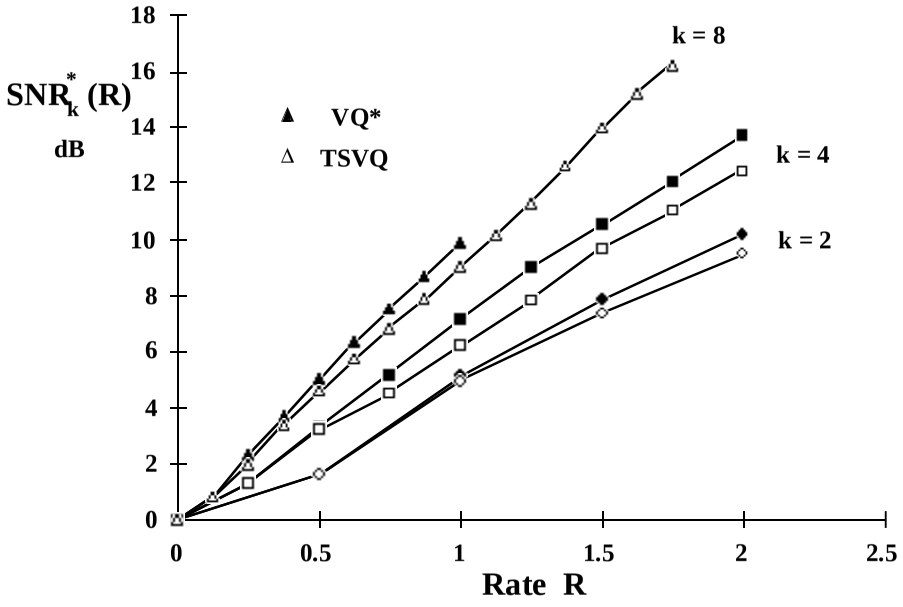


NOTES:

- The greedy algorithm does not necessarily produce the best TSVQ codes.
- A full search of the TSVQ codebook would attain lower distortion than the tree-search algorithm with which it was intended to be searched, because the tree-search does not ordinarily induce the Voronoi partition.
- The TSVQ codebook is usually not an optimal codebook for use with full search.

Performance of TSVQ & Optimum VQ

speech samples



PREDICTION OF TSVQ DISTORTION

We will substitute hypothesized point densities and inertial profiles into Bennett's integral to see what explains typical TSVQ performance.

Hypothesis 1: Point density $\lambda(\underline{x})$ is optimum (being optimistic)

Hypothesis 2: Most cells have $2k$ or fewer faces.

Why? Cells at depth d cannot have more than d faces.

However, due to the greedy design algorithm, even at depths $d >> 2k$, most cells have $2k$ or fewer faces.

Example: $k = 2$, $M = 128$, $R = 3.5$,

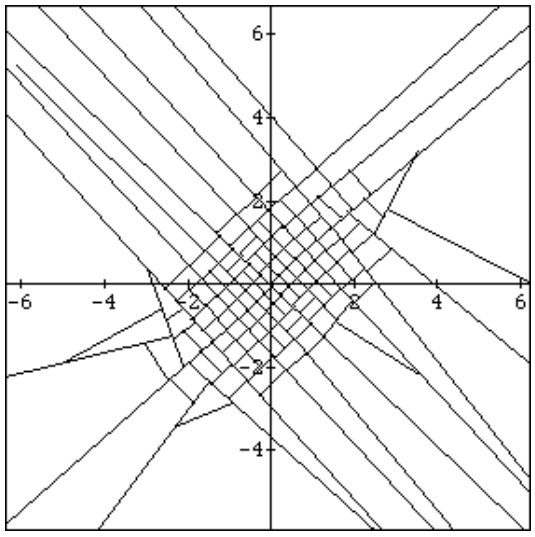
AR Gaussian source, $\rho = .9$

Observe: Cells have 2, 3 or 4 faces

Conjecture: Cubes have least NMI of polyhedra with $2k$ or fewer faces.

Hypothesis 3: Greedy design algorithm tries to form cubes. If so,

$m(\underline{x}) = \frac{1}{12}$ = NMI of cube,



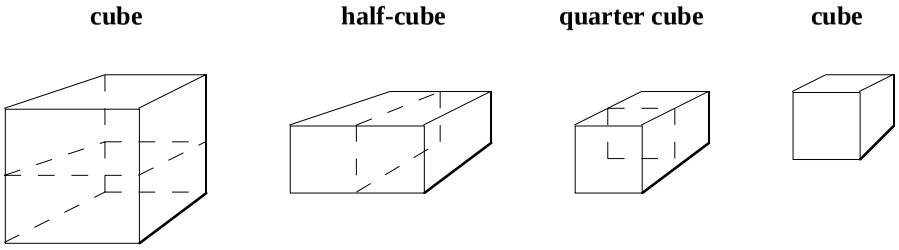
Assuming Hypotheses 1 and 3:

The loss predicted by Bennett's integral relative to optimal VQ:

.37 dB for $k = 4$

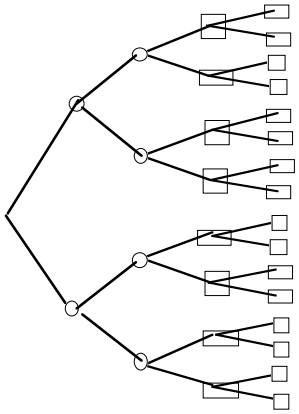
However: Greedy tree design does not produce all cubes

For example, the "child" of a cube is not a cube.



Hypothesis 3':

Cells are an equal mixture of cubes half-cubes, quarter-cubes, etc.

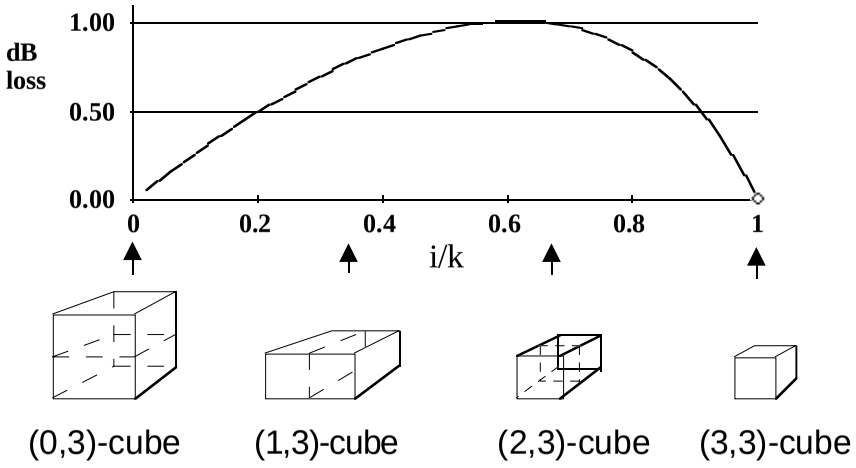


NMI of Fractional Cubes

(i,k)-cube $\triangleq$ k-dimensional cube cut in half i times

$$= \underbrace{\tfrac{1}{2}\times\tfrac{1}{2}\times\ldots\times\tfrac{1}{2}}_{i}\times\underbrace{1\times1\times\ldots\times1}_{k-i}$$

$$\frac{\text{nmi of (i,k)-cube}}{\text{nmi cube}} = \left(1-\frac{3}{4}\,\frac{i}{k}\right)4^{i/k}$$



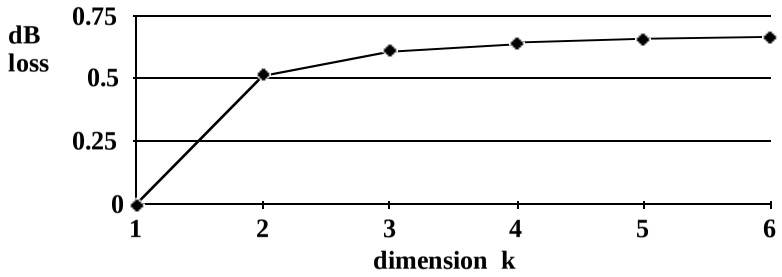
Hypothesis 3': Cells are an equal mixture of cubes half-cubes, quarter cubes, etc.

If so,

$$m(\underline{x}) = \frac{1}{12}\mu_k$$

$$\mu_k = \frac{9}{4}\,\frac{4^{1/k}}{k^2(1-4^{1/k})^2} = \frac{\text{avg. nmi fractional cubes}}{\text{nmi of cube}}$$

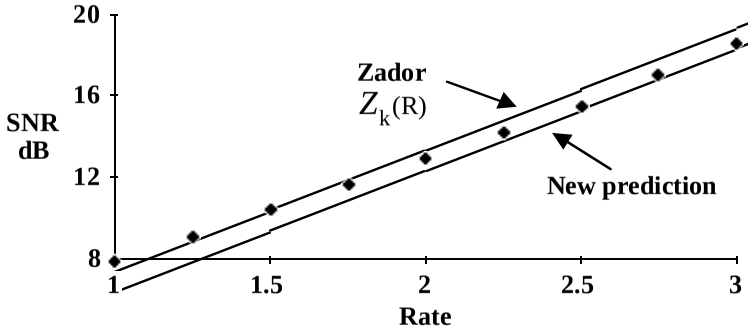
$$\rightarrow 1.17 \ (.69 \text{ dB}) \text{ as } k \rightarrow \infty$$



PREDICTION OF TSVQ DISTORTION

Hypothesis 1: Point density is optimum

Hypothesis 3': Cells are an equal mixture of cubes, half-cubes, quarter cubes, ...



k = 4, AR Gaussian source, $\rho = .8$

Prediction based on Hypotheses 1 and 3':

1.01 dB loss of SNR.
This is accurate to about .2 dB

SAMPLES OF SPEECH

TSVQ Loss = $SNR_k^*(R) - SNR_{TSVQ,k}(R)$ (dB)

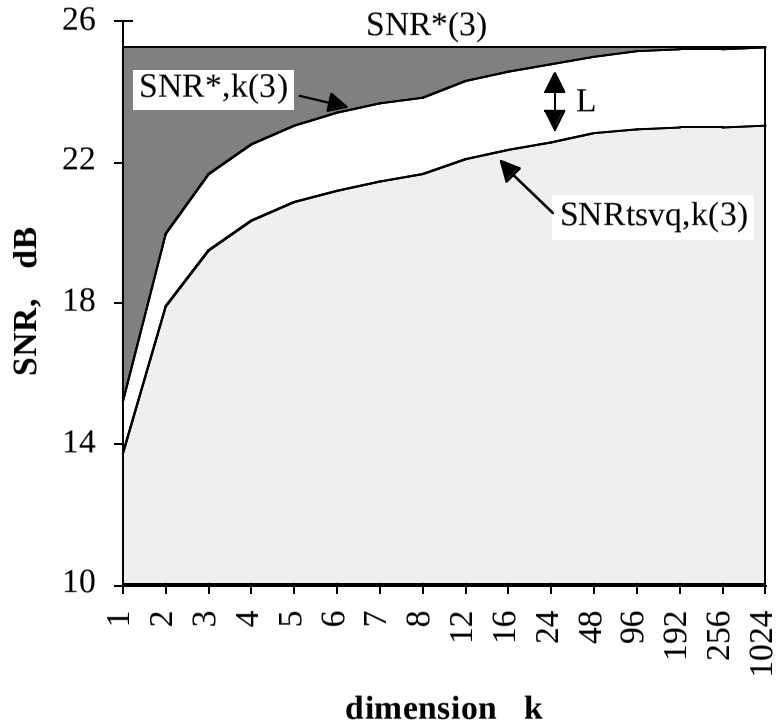
dim. k	Predicted Loss Cubic	Rect., μ_k	Total	Actual Loss	Diff.
2	.17	.51	.68	.72	.04
3	.26	.61	.86	1.01	.15
4	.37	.64	1.01	1.22	.22
∞	1.53	.69	2.22		

Rate R = 2

Conclusion:

Performance of TSVQ is accurately predicted by Hypotheses 1 & 3'.
This suggests that the greedy algorithm designs TSVQ to have, approximately, optimum point density and cells that are an equal mixture of cubes, half-cubes, etc.

EXAMPLE OF PREDICTED SNR FOR TSVQ



- R = 3
- AR Gauss Source, $\rho = .9$
- $SNR_{tsvq}(k,R)$ = $SNR(k,R) - 10 \log_{10} L$

  where $L = \frac{\mu_k/12}{M_k^*}$.

  Predictions assume equal mixture cubes, halfcubes, etc.

OTHER TREE-STRUCTURED VQ'S

- M-ary trees
  - + Complexity increases from ~2kR to ~3 $\frac{M-1}{\log_2 M}$ kR.
  - + Point density would, apparently, not be improved.
  - + Cell shapes might be improved if M is sufficiently large.
- Variable-depth trees
  - + Tree growing and pruning
  - + Rule of thumb: Grow or prune to leave only cubes at terminal nodes.