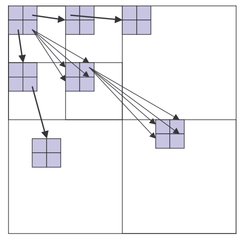
CSE 490 G
Introduction to Data Compression
Winter 2006

SPIHT
Group Testing

The Zero-Tree Method

- Invented by Shapiro, 1993, and refined by Said and Pearlman, 1996.



If a bit plane value in a low resolution subband is insignificant then it is likely that the corresponding values in higher subbands are also insignificant in the same bit plane.

Such groups of insignificant values are called zero-trees.

Zero-Tree Example

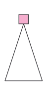
Values in a zero-tree are correlated.

Simplified SPIHT Coding

- Runs in passes - one for each bit plane.
- C[i,j] is the coefficient at index (i,j) and C[i,j,k] is the k-th bit of C[i,j].
- Encoder maintains two data structures.
  - S, a list of indices (i,j) such that C[i,j] is declared significant in the current bit plane.
  - Z, a stack of zero trees of two types.
    - rootless (R)
    - root-and-childless (RC)
  - The nodes in a zero tree are insignificant in the current bit plane. (ignore root in R and root and children in RC)

SPIHT Zero-Trees

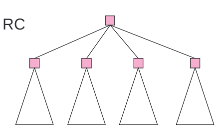
R



root is on the list S

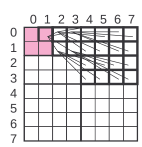
all other nodes are insignificant in current bit plane

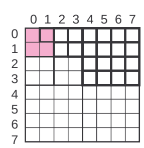
RC



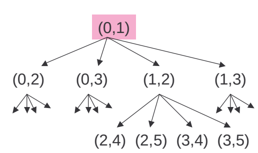
Each zero tree can be identified by its type and the index (i,j) of its root.

R-Tree Example (1)



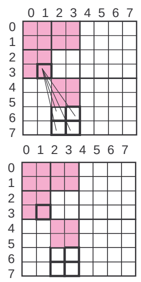


Example of zero-tree (R,0,1)

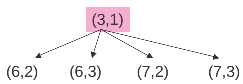


in S

R-Tree Example (2)

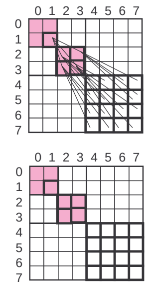


Example of zero-tree (R,3,1)

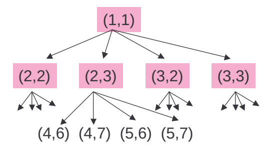


in S

RC-Tree Example



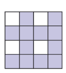
Example of zero-tree (RC,1,1)



in S

Initialization of SPIHT

- The lowest subband indices are put into S.
  - If (i,j) in lowest subband then output sign (0 for - and 1 for +) of C[i,j] and put (i,j) into S.
- A stack Z of zero trees is formed using the lowest resolution subband indices as roots.
  - If (i,j) in the lowest subband is a root of a zero tree of type R if i is odd or (i is even and j is odd).



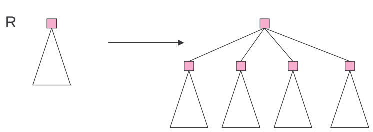
root of a zero tree

lowest subband

Iteration of SPIHT Encoder

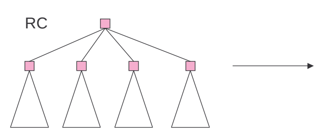
```
k-th iteration
We have list S of significant values and a stack Z of
zero trees from the previous pass or the initialization.
Significance Pass.
   while Z is not empty do
      T := pop(Z);
      if T has an index that becomes significant in bit plane k then
         output 1;
         decompose(T);
      else
         output 0;
         push T on Z'
   Z := Z'; {At this point all indices in zero trees in Z are insignificant}
Refinement Pass.
   for each (i,j) in S output the k-th significant bit, C[i,j,k].
```

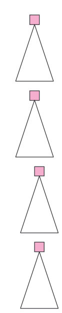

Decomposition of R



Output the sign (0 for - and 1 for +) of each of the children of the root and put them in S. Push the RC tree on the stack Z. Exception is when tree has no grandchildren. In this case, the tree dies.

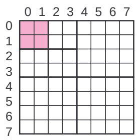
Decomposition of RC





Push each of the four trees on the stack Z.

SPIHT Coding Example: Initialization



in S

Initial data structure:

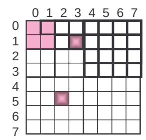
S = (0,0), (0,1), (1,0), (1,1)

Z = (R,0,1), (R,1,0), (R,1,1)

Initial output:
0 1 1 1

sign(0,0) = -
sign(0,1) = +
sign(1,0) = +
sign(1,1) = +

SPIHT Coding Example: Pass 1, Significance Pass (1)



became significant
in S

S = (0,0), (0,1), (1,0), (1,1)

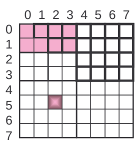
Z = (R,0,1), (R,1,0), (R,1,1)

(R,0,1) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)
output 1101 for signs of these

Z = (RC,0,1), (R,1,0), (R,1,1)

SPIHT Coding Example: Pass 1, Significance Pass (2)



became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)

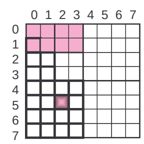
Z = (RC,0,1), (R,1,0), (R,1,1)

(RC,0,1) is not significant
output 0

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)

Z = (R,1,0), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (3)



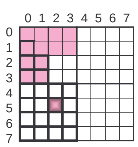
became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)
Z = (R,1,0), (R,1,1)
Z' = (RC,0,1)

(R,1,0) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
output 1100 for signs of these
Z = (RC,1,0), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (4)



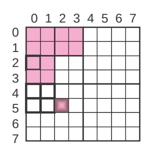
became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (RC,1,0), (R,1,1)
Z' = (RC,0,1)

(RC,1,0) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,0), (R,2,1), (R,3,0),
(R,3,1), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (5)

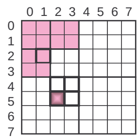


became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,0), (R,2,1), (R,3,0),
(R,3,1) (R,1,1)
Z' = (RC,0,1)

(R,2,0) is not significant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,1), (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (6)



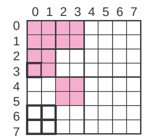
became significant

in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,1), (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

(R,2,1) is significant
output 1
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
output 1010 for signs of these
Z = (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (7)



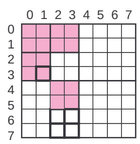
became significant

in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

(R,3,0) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,1), (R,1,1)
Z' = (R,3,0), (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (8)



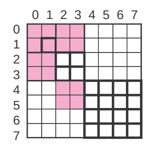
became significant

in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,1), (R,1,1)
Z' = (R,3,0), (R,2,0),(RC,0,1)

(R,3,1) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1)
Z' = (R,3,1), (R,3,0), (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (9)

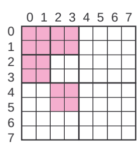


became significant

in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1)
Z' = (R,3,1), (R,3,0), (R,2,0),(RC,0,1)
(R,1,1) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z =
Z' = (R,1,1), (R,3,1), (R,3,0),
(R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Refinement Step



in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1), (R,3,1), (R,3,0),
(R,2,0),(RC,0,1)

output 1011000100000010
one bit for each member of S.

37 total bits in pass 1 were output.
Initialization was 4 bits.
Total of 41 bits to send 64 bits plus
16 sign bits.

SPIHT Decoding

- The decoder emulates the encoder.
  - The decoder maintains exactly the same data structures as the encoder.
  - When the decoder has popped the Z stack to examine a zero tree it receives a bit telling it whether the tree is significant. The decoder can then do the right thing.
    - If it is significant then it does the decomposition.
    - If it is not significant then it deduces a number of zeros in the current bit plane.

SPIHT Decoder

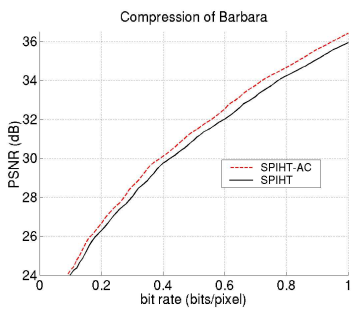
<u>k-th iteration</u>
We have list S of significant values and a stack Z of zero trees from the previous pass or the initialization.
<u>Significance Pass</u>.
```
   while Z is not empty do
      T := pop(Z);
      input := read;
      if input = 1 then decompose(T);
      else push T on Z'
   Z := Z'; {At this point all indices in zero trees in Z are insignificant}
```
<u>Refinement Pass</u>.
```
   for each (i,j) in S do C[i,j,k] := read.
```

In decompose the signs of coefficients are input

Notes on SPIHT

- SPIHT was very influential
  - People really came to believe that wavelet compression can really be practical (fast and effective).
- To yield the best compression an arithmetic coding step is added to SPIHT
  - The improvement is about .5 DB

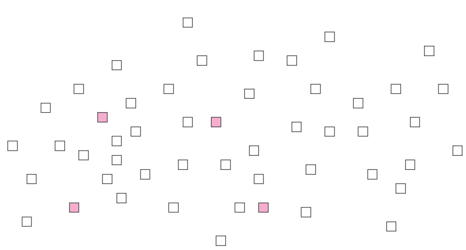
SPIHT-AC



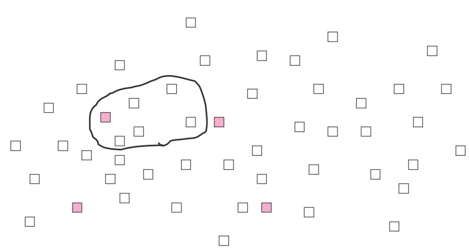
Group Testing (Dorfman 1943)

- Given n items, with s items significant
- Use group tests to identify significant items
  - Group test of size k
    - Group is insignificant: all k items insignificant
    - Group is significant: at least one significant
- Goal: Minimize number of group tests

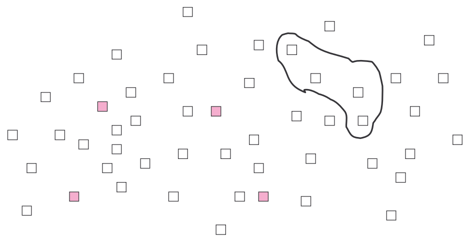
Group Testing (Dorfman 1943)



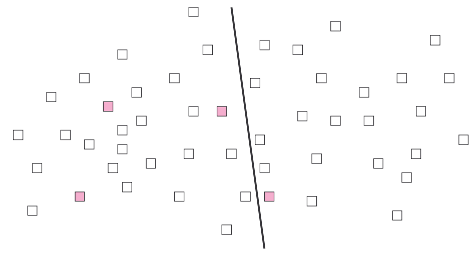
Group Testing (Dorfman 1943)



Group Testing (Dorfman 1943)



Group Testing (Dorfman 1943)



classes to reduce group tests

Zerotree Coding as Group Testing

- Coefficients = items
- Testing trees for significance = group test
- Zerotree coding = one particular group testing algorithm
- Zerotree coding & group testing have similar goals

Hwang's Group Testing Algorithm (1972)

- Repeat a Group Iteration until all significant items are found
- Group Interation
  - Test group G containing k unidentified items
  - If G is significant, find a significant item in $\log_2 k$ tests
    - Each subsequent test is a subset of G
    - Size of test group is halved each time

Group Iteration of size 8

	Group Test Result	Code
$\boxed{\texttt{? ? ? ? ? ? ? ?}}$	Insignificant	0
I I I I I I I I		

Group Iteration of size 8

	Group Test Result	Code
$\boxed{\texttt{? ? ? ? ? ? ? ?}}$	Significant	1
$\boxed{\texttt{? ? ? ?}}$? ? ? ?	Insignificant	0
I I I I $\boxed{\texttt{? ?}}$? ?	Significant	1
I I I I $\boxed{\texttt{?}}$? ? ?	Insignificant	0
I I I I I S ? ?		

- Equivalent to elementary Golomb code of order 8

Group Testing for Wavelet Image Coding (GTW)

- Hong and Ladner (2000)
- New method for encoding significance pass:
  - Uses Hwang's Group Testing Algorithm
  - Divide wavelet coefficients into classes
  - Every group test performed is on coefficients in the same class

GTW Significance Pass Overview

- Repeat until all coefficients are coded
  - Pick a set of coefficients from the class that is most likely to have significant coefficients.
  - Do one group iteration on the set. The group size is determined by the adaptive group tester.
    - Output the results of the group tests
  - If a significant coefficient is found then
    - Output its sign
    - Update classes of neighboring coefficients

GTW's adaptive group tester

- Choosing group iteration size k:
  - Ramp up: start with k=1
    - While group insignificant, double k
  - Steady state: use past history to estimate probability p of insignificance
    - Optimal k using Gallager & Van Voorhis' (1975) result

$$k = \left\lceil \frac{-1}{\log_2 p} \right\rceil$$

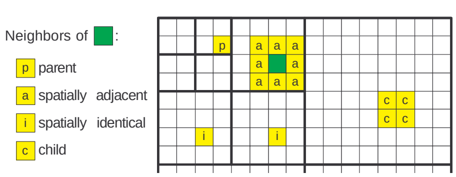
Same as the adaptive Golomb coding algorithm.

GTW Classes

- Coefficients with similar characteristics put into the same class
- Classes are ordered so that classes with coefficients more likely to be significant are tested first.
- Class characteristics
  - Significant neighbor count
  - pattern type
  - Subband level

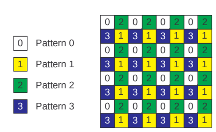
Significant Neighbor Metric

- Count # of significant neighbors
- Example Neighborhood

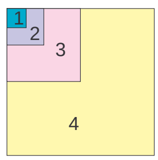


Pattern Type

- Accounts for correlation exists between neighbors
  - Example pattern types for a subband



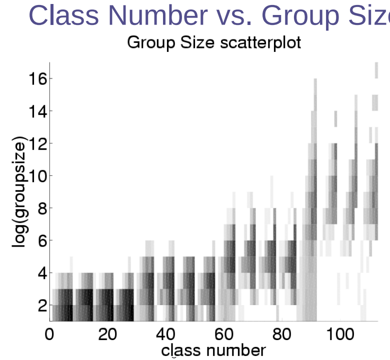
Subband Level



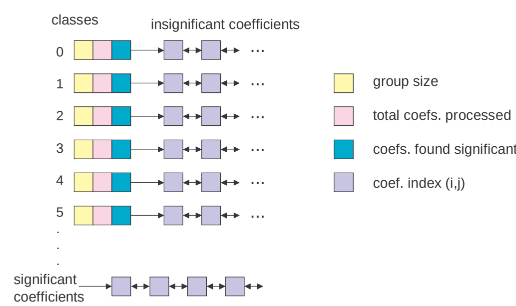
Class Ordering

- Significant neighbor count
  - children count for at most 1
  - 0,1,2 or 3 or more
- Pattern Types 4 and Subband levels 7
- Total number of classes 112
- Ordering
  - First by significant neighbor count (large to small)
  - Second by pattern type (small to large)
  - Third by subband level (small to large)

Class Number vs. Group Size



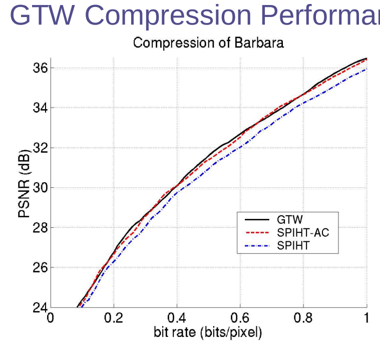
GTW Data Structures



Decoding

- Decoding algorithm is identical to the encoding algorithm except
  - Decoder knows the results of group test from the compressed bit stream

GTW Compression Performance



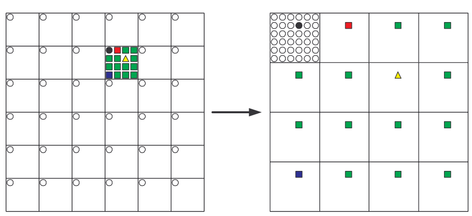
Flexibility of Group Testing

- Flexibility
  - Significant data sent first
  - Classes defined to focus of significant data
  - Data can move from class to class
  - We always get a progressive coder
- Applications
  - DCT
  - Lapped Transforms
  - Wavelet Packets

GT-DCT

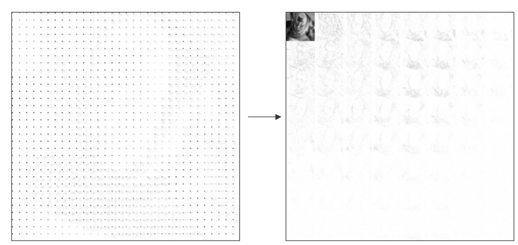
- Hong, Ladner, Riskin (2001)
- Group testing for the discrete cosine transform.
- We do bit-plane coding of the DCT coefficients.
- DCT classes are defined.
- Group testing done first on the classes that have the smallest group size.

Reorganizing Block Transform Coefficients



block organization subband organization

Reorganizing Block Transform Coefficients



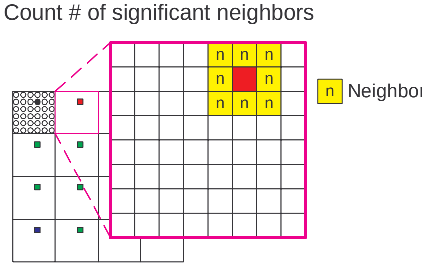
block organization subband organization

GT-DCT Classes

- Based on subband reorganization of coefficients
- Class characteristics
  - Significant neighbor metric
  - Subband level

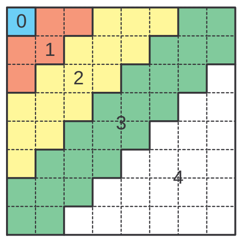
Significant Neighbor Metric

- Count # of significant neighbors



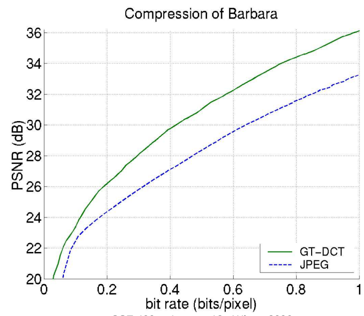
subband organization

Subband Level



subband organization

GT-DCT vs JPEG



Group Testing Notes

- Group testing provides a unified and flexible way to approach bit-plane coding of transformed images.
  - Need good classes (contexts)
  - Need good group testing algorithms (adaptive Golomb coding works)
- Compression performance is outstanding
- Group testing is quite a bit more time consuming than JPEG and SPIHT.
  - need some good data structures and engineering