15-750:Algorithms in the Real World

Transformation Techniques

Why transform?

- Help skew the probabilities
  - Why?
  - Recall higher the skew easier it is to compress

- In many algorithms message sequences are transformed into **integers with a skew towards small integers**

- We will take a detour to study codes for integers ...

Integer codes

- There are several "fixed" codes for encoding natural numbers
- With non-decreasing codeword lengths

Integer codes: binary

n	Binary
1	..001
2	..010
3	..011
4	..100
5	..101
6	..110

"Minimal" binary representation: Drop leading zeros

Q: What is the problem with minimal binary representation?

Not a prefix code!

Integer codes: Unary

n	Binary	Unary
1	..001	0
2	..010	10
3	..011	110
4	..100	1110
5	..101	11110
6	..110	111110

n represented as $(n-1)$ 1's and one 0: (0's and 1's can be interchanged)

Q: For what probability distribution unary codes are optimal prefix codes?

<Better code in HW>

Transformation Techniques

1. Run length coding
2. Move-to-front coding
3. Residual coding
4. Burrows-Wheeler transform
5. Linear transform coding

1. Run Length Coding

Code by specifying message value followed by the number of repeated values:

e.g. **abbbaacccca => (a,1),(b,3),(a,2),(c,4),(a,1)**

The characters and counts can be coded based on frequency (i.e., probability coding).

Typically low counts such as 1 and 2 are more common => use small number of bits overhead for these.

Used as a sub-step in many compression algorithms.

2. Move to Front (MTF) Coding

- Transforms message sequence into sequence of integers
- Then probability code

Start with values in a total order: e.g.: [a,b,c,d,…]

For each message

- output the position in the order
- move to the front of the order.

e.g.: **c a**

c => output: 3, new order: [c,a,b,d,e,…]
a => output: 2, new order: [a,c,b,d,e,…]

Probability code the output.

2. Move to Front (MTF) Coding

The hope is that there is a bias for small numbers.

Q: Why?

Temporal locality

Takes advantage of **temporal locality**

Used as a sub-step in many compression algorithms.

3. Residual Coding

Typically used for message values that represent some sort of amplitude:
e.g. gray-level in an image, or amplitude in audio.

Basic Idea:

- Guess next value based on current context.
- Output difference between guess and actual value.
- Use probability code on the output.

E.g.: Consider compressing a stock value over time.

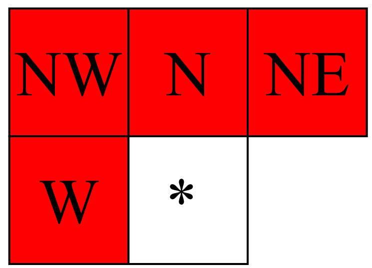
Residual coding is used in JPEG Lossless

Use of residual coding in JPEG-LS

JPEG Lossless

Codes in Raster Order.

Uses 4 pixels as context:



Tries to guess value of * based on W, NW, N and NE.

The residual between guessed and actual value is found and then coded using a Golomb-like code.

(Golomb codes are similar to Gamma codes)

4. Burrows –Wheeler Transform

When used for file compression: Breaks file into fixed-size blocks and encodes each block separately.

For each block:

- BWT generates a new string which is a permutation of the characters in the original string
- First will describe intuitively what’s happening and then we will see efficient computation of BWT

4. Burrows –Wheeler Transform

- Assume the string S ends with a special, unique character $
- List all cyclic rotations of S
- Lexicographically sort them
- This give BWT matrix
- BWT(S) = last column of the BWT matrix

E.g. S= decode

BWT matrix =

BWT(S) = last column = eeo$ddc

Workout Example

Q: Why is the output more easier to compress?
(Tends to group same characters together.. Why?)

Can we invert BW Transform?

BWT Output: Last column of the BWT matrix (L)

Workout
Example

How can we get the first column (F)
from the output column (L)?

Sort!

Any problem? Equal valued chars

Burrows-Wheeler (Continued)

Theorem: (informal statement) In the BWT matrix, equal valued characters appear in the same order in the last column (L) as in the first column (F)

Proof sketch:

- In F (the first column), equal valued chars all appear together and are ordered by their suffixes (right context).
- In L (the last column), equal valued chars can be scattered, but their relative ordering is still sorted based on the same suffixes (right context) due to the cyclical rotations.

Workout Example

BWT and suffix arrays

- The naïve way of generating BWT: O($n^2 \log n$)
  - Sorting n length strings (each comparison takes O(n))
- Can use Suffix arrays instead to construct BWT in O(n)
  - If you delete the characters after $ they are are precisely suffixes
- How to get BWT since no last column?
  - -1 of that indexes to the first column elements
- Inverting:
  - Several optimizations to speed up inverting BWT exist
  - We won't have time to cover them

BZIP

Transform 1: (Burrows Wheeler)

- **input** : character string (block)
- **output** : reordered character string

Transform 2: (move to front)

- **input** : character string
- **output** : MTF numbering

Transform 3: (run length)

- **input** : MTF numbering
- **output** : sequence of run lengths

Probabilities: (on run lengths)

Dynamic based on counts for each block.

Coding: Originally arithmetic, but changed to Huffman in bzip2 due to patent concerns

15-750:Algorithms in the Real World

Linear Transform Coding

(for both lossless and lossy compression)

5. Linear Transform Coding

Goal: Transform the data into a form that is easily compressible (through **lossless** or **lossy** compression)

Select a set of linear basis functions ϕ_i that span the space

- sin, cos, spherical harmonics, wavelets, …

How to Pick a Transform

Goals:

- Decorrelate the data
- Low coefficients for many terms
- Basis functions that can be ignored from the perception point-of-view