15-853: Algorithms in the Real World

Data Compression IV.5

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...
Information Theory: Entropy, etc.
Probability Coding: Huffman + Arithmetic Coding
Applications of Probability Coding: PPM + others
Lempel-Ziv Algorithms: LZ77, gzip, compress, ...
Other Lossless Algorithms: Burrows-Wheeler
Lossy algorithms for images: JPEG, MPEG, ...
- Scalar and vector quantization
- JPEG and MPEG
Compressing graphs and meshes: BBK

Scalar Quantization

Quantize regions of values into a single value:

output
output

uniform
non uniform

Can be used to reduce # of bits for a pixel

Vector Quantization

In

Generate Vector

Find closest code vector

Codebook Index

Encode

Index

Generate Output

Decode

Out
Vector Quantization

What do we use as vectors?
- Color (Red, Green, Blue)
  - Can be used, for example to reduce 24 bits/pixel to 8 bits/pixel
  - Used in some terminals to reduce data rate from the CPU (colormaps)
- K consecutive samples in audio
- Block of K pixels in an image
How do we decide on a codebook
- Typically done with clustering

Linear Transform Coding

Want to encode values over a region of time or space
- Typically used for images or audio
Select a set of linear basis functions \( \phi \), that span the space
- \( \sin, \cos, \) spherical harmonics, wavelets, ...
- Defined at discrete points

Vector Quantization: Example

Linear Transform Coding

Coefficients: \( \Theta_i = \sum_j x_j \phi_i(j) = \sum_j x_j a_{ij} \)
\( \Theta_i = i^{th} \) resulting coefficient
\( x_j = j^{th} \) input value
\( a_{ij} = ij^{th} \) transform coefficient = \( \phi_i(j) \)

In matrix notation: \( \Theta = Ax \)
\( x = A^{-1} \Theta \)
Where \( A \) is an \( n \times n \) matrix, and each row defines a basis function
**Example: Cosine Transform**

\[ \Theta_j = \sum_j x_j \phi_j(j) \]

**Other Transforms**

Polynomial:

![Polynomial](image)

Wavelet (Haar):

![Wavelet](image)

**How to Pick a Transform**

**Goals:**
- Decorrelate
- Low coefficients for many terms
- Basis functions that can be ignored by perception

Why is using a Cosine of Fourier transform across a whole image bad?

How might we fix this?

**Usefulness of Transform**

Typically transforms A are orthonormal: \( A^\dagger = A^T \)

**Properties of orthonormal transforms:**

\[ \sum x^2 = \sum \Theta^2 \] (energy conservation)

Would like to compact energy into as few coefficients as possible

\[ G_{tc} = \frac{1}{n} \sum \sigma_i^2 \left( \prod \sigma_i^2 \right)^{1/n} \] (the transform coding gain)

\[ \sigma_i = (\Theta_i - \Theta_{i-1}) \]

The higher the gain, the better the compression
Case Study: JPEG

A nice example since it uses many techniques:
- Transform coding (Cosine transform)
- Scalar quantization
- Difference coding
- Run-length coding
- Huffman or arithmetic coding

JPEG (Joint Photographic Experts Group) was designed in 1991 for lossy and lossless compression of color or grayscale images. The lossless version is rarely used.
Can be adjusted for compression ratio (typically 10:1)

JPEG: Quantization Table

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>11</td>
<td>10</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>51</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>58</td>
<td>60</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>57</td>
<td>69</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>22</td>
<td>29</td>
<td>51</td>
<td>87</td>
<td>80</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>37</td>
<td>56</td>
<td>68</td>
<td>109</td>
<td>103</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>35</td>
<td>55</td>
<td>64</td>
<td>81</td>
<td>104</td>
<td>113</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>64</td>
<td>78</td>
<td>87</td>
<td>103</td>
<td>121</td>
<td>120</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>92</td>
<td>95</td>
<td>98</td>
<td>112</td>
<td>100</td>
<td>103</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

Also divided through uniformaly by a quality factor which is under control.

JPEG: Block scanning order

Uses run-length coding for sequences of zeros
**JPEG: example**

.125 bits/pixel (factor of 200)

---

**Case Study: MPEG**

Pretty much JPEG with **interframe coding**

Three types of frames
- I = intra frame (aprox. JPEG) anchors
- P = predictive coded frames
- B = bidirectionally predictive coded frames

Example:
- Type: I B B P B B P B P B B I
- Order: 1 3 4 2 6 7 5 9 10 8 12 13 11

I frames are used for random access.

---

**MPEG matching between frames**

---

**MPEG: Compression Ratio**

356 x 240 image

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>18KB</td>
<td>7/1</td>
</tr>
<tr>
<td>P</td>
<td>6KB</td>
<td>20/1</td>
</tr>
<tr>
<td>B</td>
<td>2.5KB</td>
<td>50/1</td>
</tr>
<tr>
<td>Average</td>
<td>4.8KB</td>
<td>27/1</td>
</tr>
</tbody>
</table>

30 frames/sec x 4.8KB/frame x 8 bits/byte = 1.2 Mbits/sec + .25 Mbits/sec (stereo audio)

HDTV has 15x more pixels = 18 Mbits/sec
**MPEG in the “real world”**

- DVDs
  - Adds “encryption” and error correcting codes
- Direct broadcast satellite
- HDTV standard
  - Adds error correcting code on top
- Storage Tech “Media Vault”
  - Stores 25,000 movies

Encoding is much more expensive than decoding.
Still requires special purpose hardware for high resolution and good compression.

---

**Wavelet Compression**

- A set of localized basis functions
- Avoids the need to block

**mother function** $\phi(x)$

$$\phi_s(x) = \phi(2^s x - l)$$

$s =$ scale $l =$ location

**Requirements**

$$\int_{-\infty}^{\infty} \psi(x) dx = 0$$ and $$\int_{-\infty}^{\infty} |\psi(x)|^2 dx < -\infty$$

Many mother functions have been suggested.

---

**Haar Wavelets**

Most described, least used:

$$\phi(x) = \begin{cases} 
1 & 0 \leq x < 1/2 \\
-1 & 1/2 \leq x < 1 \\
0 & \text{otherwise}
\end{cases}$$

$$H_s(x) = \phi(2^s x - l)$$

---

**Haar Wavelet in 2d**
Discrete Haar Wavelet Transform

How do we convert this to the wavelet coefficients?

Haar Wavelet Transform: example

a = 2  1  2 -1 -2  0  2 -2
    =  1.5 0.5 -1  0  0.5 1.5 -1  2
    =  1  -0.5 0.5 -0.5
    =  0.25 0.75 0.5 0.5 0.5 1.5 -1  2

Averages
Differences
Linear time!

Wavelet decomposition
**Morlet Wavelet**

\[ \phi(x) = \text{Gaussian \cdot Cosine} = e^{-\left(\frac{x^2}{2}\right)} \cos(5x) \]

Corresponds to wavepackets in physics.

**Daubechies Wavelet**

![Daubechies Wavelet Graph]

**JPEG2000**

**Overall Goals:**
- High compression efficiency with good quality at compression ratios of .25bpp
- Handle large images (up to \(2^{32} \times 2^{32}\))
- Progressive image transmission
  - Quality, resolution or region of interest
- Fast access to various points in compressed stream
- Pan and Zoom while only decompressing parts
- Error resilience

**JPEG2000: Outline**

Main similarities with JPEG
- Separates into Y, I, Q color planes, and can downsample the I and Q planes
- Transform coding

Main differences with JPEG
- Wavelet transform
  - Daubechies 9-tap/7-tap (irreversible)
  - Daubechies 5-tap/3-tap (reversible)
- Many levels of hierarchy (resolution and spatial)
- Only arithmetic coding
**JPEG2000: 5-tap/3-tap**

\[ h[i] = a[2i-1] - (a[2i] + a[2i-2])/2; \]
\[ l[i] = a[2i] + (h[i-1] + h[i] + 2)/2; \]

- \( h[i] \): is the "high pass" filter, i.e., the differences it depends on 3 values from a (3-tap)
- \( l[i] \): is the "low pass" filter, i.e., the averages it depends on 5 values from a (5-tap)

Need to deal with boundary effects.
This is reversible: assignment

---

**JPEG 2000: Outline**

- **A spatial and resolution hierarchy**
- **Tiles**: Makes it easy to decode sections of an image. For our purposes we can imagine the whole image as one tile.
- **Resolution Levels**: These are based on the wavelet transform. High-detail vs. Low detail.
- **Precinct Partitions**: Used within each resolution level to represent a region of space.
- **Code Blocks**: blocks within a precinct
- **Bit Planes**: ordering of significance of the bits

---

**JPEG2000: Precincts**

![Precinct Diagram]

---

**JPEG vs. JPEG2000**

- JPEG: .125bpp
- JPEG2000: .125bpp
**Compression Outline**

**Introduction:** Lossy vs. Lossless, Benchmarks, ...
**Information Theory:** Entropy, etc.
**Probability Coding:** Huffman + Arithmetic Coding
**Applications of Probability Coding:** PPM + others
**Lempel-Ziv Algorithms:** LZ77, gzip, compress, ...
**Other Lossless Algorithms:** Burrows-Wheeler
**Lossy algorithms for images:** JPEG, MPEG, ...
**Compressing graphs and meshes:** BBK

---

**Compressing Structured Data**

So far we have concentrated on Text and Images, compressing sound is also well understood.
What about various forms of "structured" data?
- Web indexes
- Triangulated meshes used in graphics
- Maps (mapquest on a palm)
- XML
- Databases

---

**Compressing Graphs**

**Goal:** To represent large graphs compactly while supporting queries efficiently
- e.g., adjacency and neighbor queries

**Applications:**
- Large web graphs
- Large meshes
- Phone call graphs

---

**Results**

O(n)-bit compression on "separable graphs" with n vertices, with O(1) access time
Experimental results:
- 6-12 bits/edge in total
- access time faster than linked-list representation
**Vertex Separators**

A vertex separator for $G=(V,E)$ is a set of vertices $U \subset V$, that partitions $V/U$ into two components $V_1$ and $V_2$.

A class of graphs $S$ satisfies a $f(n)$-vertex separator theorem if

\[ \exists \alpha < 1, \beta > 0 \]
\[ \forall (V,E) \in S, \exists \text{ separator } U, \]
\[ |U| < \beta f(|V|), \]
\[ |V_i| < \alpha |V|, i = 1,2 \]

A separable class of graphs is one that satisfies an $n^{1/2}$-separator theorem, $c<1$.  

**Edge Separators**

An edge separator for $(V,E)$ is a set of edges $E' \subset E$ whose removal partitions $V$ into two components $V_1$ and $V_2$.

A class of graphs $S$ satisfies a $f(n)$-edge separator theorem if

\[ \exists \alpha < 1, \beta > 0 \]
\[ \forall (V,E) \in S, \exists \text{ separator } E', \]
\[ |E'| < \beta f(|V|), \]
\[ |V_i| < \alpha |V|, i = 1,2 \]

Any class of graphs that satisfies an edge separator theorem satisfies the corresponding vertex separator theorem as well.

**Separable Classes of Graphs**

Planar graphs: $O(n^{1/2})$ separators

Well-shaped meshes in $\mathbb{R}^2$: $O(n^{1/2})$ [Miller et al.]

Nearest-neighbor graphs

In practice, good separators from circuit graphs, street graphs, web connectivity graphs, router connectivity graphs

Note: All separable classes of graphs have bounded density ($m = O(n)$)

**Main Ideas**

For good edge separators
- Number vertices using separators
- Use difference coding on adjacency lists
- Using efficient data structure for indexing

For good vertex separators
- Each vertex assigned multiple labels
- Separate "root-find" data structure to map labels to a representative
- For adjacency queries, may need to direct graph
Compressed Adjacency Tables

<table>
<thead>
<tr>
<th>#</th>
<th>D</th>
<th>Neighbors</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3 4</td>
<td>3 1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5 6 8</td>
<td>3 1 1 2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2 7</td>
<td>-3 7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1 7 8</td>
<td>-4 1 0 1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1 6 8</td>
<td>-4 5 2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1 2 5</td>
<td>-5 1 3</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3 4 8</td>
<td>-4 1 4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1 4 5 7</td>
<td>-7 3 1 2</td>
</tr>
</tbody>
</table>

Compressed Adjacency Tables

<table>
<thead>
<tr>
<th>#</th>
<th>D</th>
<th>Neighbors</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0 2 3</td>
<td>-1 2 1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1 3 4</td>
<td>-1 2 1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1 2 4 5</td>
<td>-1 2 2 1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2 3 5 6</td>
<td>-2 1 2 1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3 4 6 7</td>
<td>-2 1 2 1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4 5 8</td>
<td>-2 1 3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>5 8</td>
<td>-2 3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6 7</td>
<td>-2 1</td>
</tr>
</tbody>
</table>

Log-sized Codes

Log-sized code: Any prefix code that takes $O(\log(d))$ bits to represent an integer $d$.
Gamma code, delta code, skewed Bernoulli code

Example: Gamma code
Prefix: unary code for $\lfloor \log d \rfloor$
Suffix: binary code for $d-2^{\lfloor \log d \rfloor}$
(binary code for $d$, except leading 1 is implied)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
</tr>
<tr>
<td>4</td>
<td>001101</td>
</tr>
<tr>
<td>5</td>
<td>001101</td>
</tr>
<tr>
<td>6</td>
<td>00110</td>
</tr>
<tr>
<td>7</td>
<td>00111</td>
</tr>
<tr>
<td>8</td>
<td>0001000</td>
</tr>
</tbody>
</table>

Difference Coding

For each vertex, encode:
- Degree
- Sign of first entry
- Differences in adjacency list

Concatenate vertex encodings to encode the graph

<table>
<thead>
<tr>
<th># D Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 3 1</td>
</tr>
<tr>
<td>0 11 1</td>
</tr>
<tr>
<td>0001 0010 00110 1</td>
</tr>
</tbody>
</table>

15-833 Page 45

15-833 Page 46

15-833 Page 47

15-833 Page 48
**Indexing**

**Problem:** We need to find the start of a vertex quickly.

**Formally:** Given \( n \) numbers in range \( 1..O(n) \), prepare a data structure that returns the \( k^{th} \) smallest number.

Can be supported with \( O(n) \) bits and \( O(1) \) access time.

---

**Renumbering with Edge Separators**

---

**Renumbering with Edge Separators**

---

**Renumbering with Edge Separators**
Renumbering with Edge Separators

Proof Outline
Bound cost of adjacency lists: cost of edge (a,b) is at most $O(\log(|a-b|))$
If an edge is a separator in a graph of size $d$, then its cost is at most $\log(d)$

$n^2$ edges in separator: cost $n^2 \log(n)$

$T(n) = n^2 \log(n) + 2T(n/2)$

$= O(n)$

Theorem (edge separators)
Any class of graphs that allows $O(n^2)$ edge separators can be compressed to $O(n)$ bits with $O(1)$ access time using:
- Difference coded adjacency lists
- $O(n)$-bit indexing structure

Experimental Results: Test Graphs

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vtxs</th>
<th>Edges</th>
<th>Max Degree</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>448695</td>
<td>3314611</td>
<td>37</td>
<td>3D mesh [28]</td>
</tr>
<tr>
<td>feocean</td>
<td>143437</td>
<td>409593</td>
<td>6</td>
<td>3D mesh [28]</td>
</tr>
<tr>
<td>m14b</td>
<td>214765</td>
<td>1679018</td>
<td>40</td>
<td>3D mesh [28]</td>
</tr>
<tr>
<td>ibm17</td>
<td>185405</td>
<td>2235716</td>
<td>150</td>
<td>circuit [2]</td>
</tr>
<tr>
<td>ibm18</td>
<td>210613</td>
<td>2221860</td>
<td>173</td>
<td>circuit [2]</td>
</tr>
<tr>
<td>CA</td>
<td>1971281</td>
<td>2766667</td>
<td>12</td>
<td>street map [27]</td>
</tr>
<tr>
<td>PA</td>
<td>1090920</td>
<td>1541898</td>
<td>9</td>
<td>street map [27]</td>
</tr>
<tr>
<td>google</td>
<td>916428</td>
<td>5105039</td>
<td>6326</td>
<td>web links [9]</td>
</tr>
<tr>
<td>googleO</td>
<td>916428</td>
<td>5105039</td>
<td>456</td>
<td>web links [9]</td>
</tr>
<tr>
<td>pacnet</td>
<td>112900</td>
<td>181620</td>
<td>429</td>
<td>routers [24]</td>
</tr>
<tr>
<td>scan</td>
<td>228298</td>
<td>320168</td>
<td>1037</td>
<td>routers [24]</td>
</tr>
</tbody>
</table>
Performance: Adjacency Table

<table>
<thead>
<tr>
<th></th>
<th>cdf</th>
<th>metric-cf</th>
<th>bu-bpq</th>
<th>bu-cf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>T/Pc</td>
<td>T/Pc</td>
<td>T/Pc</td>
<td>T/Pc</td>
</tr>
<tr>
<td>auto</td>
<td>0.79</td>
<td>9.88</td>
<td>15.3</td>
<td>5.17</td>
</tr>
<tr>
<td>feclean</td>
<td>0.06</td>
<td>13.88</td>
<td>388.88</td>
<td>7.66</td>
</tr>
<tr>
<td>radh</td>
<td>0.31</td>
<td>10.66</td>
<td>181.41</td>
<td>4.81</td>
</tr>
<tr>
<td>ibm17</td>
<td>0.44</td>
<td>13.03</td>
<td>136.48</td>
<td>6.18</td>
</tr>
<tr>
<td>ibm18</td>
<td>0.48</td>
<td>11.88</td>
<td>135.22</td>
<td>5.77</td>
</tr>
<tr>
<td>CA</td>
<td>0.76</td>
<td>8.41</td>
<td>362.67</td>
<td>4.58</td>
</tr>
<tr>
<td>PA</td>
<td>0.43</td>
<td>8.47</td>
<td>304.06</td>
<td>4.45</td>
</tr>
<tr>
<td>google</td>
<td>1.4</td>
<td>7.44</td>
<td>190.91</td>
<td>4.68</td>
</tr>
<tr>
<td>googleO</td>
<td>1.4</td>
<td>11.08</td>
<td>181.6</td>
<td>6.78</td>
</tr>
<tr>
<td>lucent</td>
<td>0.04</td>
<td>7.56</td>
<td>390.75</td>
<td>5.62</td>
</tr>
<tr>
<td>scan</td>
<td>0.12</td>
<td>8.00</td>
<td>280.25</td>
<td>5.64</td>
</tr>
</tbody>
</table>

Avg | 10.02 | 252.78 | 5.62 | 13.85 | 5.86 | 34.54 | 5.56 |

Time is to create the structure, normalized to time for DFS.

Performance: Overall

| Graph | Array | | List | | bu-cf/semi | |
|-------|-------|-------|-------|-------|-------|
|       | time | space | time | space | time | space |
| auto | 0.24 | 34.2 | 0.61 | 66.2 | 0.51 | 7.17 |
| feclean | 0.04 | 37.6 | 0.08 | 69.6 | 0.09 | 11.75 |
| m14b | 0.11 | 34.1 | 0.29 | 66.1 | 0.24 | 6.7 |
| ibm17 | 0.15 | 33.3 | 0.40 | 65.3 | 0.34 | 7.72 |
| ibm18 | 0.14 | 33.5 | 0.38 | 65.6 | 0.32 | 7.33 |
| CA | 0.34 | 43.4 | 0.56 | 75.4 | 0.58 | 11.66 |
| PA | 0.19 | 43.3 | 0.31 | 75.3 | 0.32 | 11.68 |
| google | 0.24 | 37.7 | 0.49 | 69.7 | 0.45 | 7.86 |
| googleO | 0.24 | 37.7 | 0.50 | 69.7 | 0.51 | 9.90 |
| lucent | 0.02 | 42.0 | 0.04 | 74.0 | 0.05 | 11.87 |
| scan | 0.04 | 43.4 | 0.06 | 75.4 | 0.08 | 12.85 |

Page 58

Conclusions

O(n)-bit representation of separable graphs with O(1)-time queries
Space efficient and fast in practice for a wide variety of graphs.

Compress Summary

Compression is all about probabilities

We want the model to skew the probabilities as much as possible (i.e., decrease the entropy)
Compression Summary

How do we figure out the probabilities
  - Transformations that skew them
    • Guess value and code difference
    • Move to front for temporal locality
    • Run-length
    • Linear transforms (Cosine, Wavelet)
    • Renumber (graph compression)
  - Conditional probabilities
    • Neighboring context
In practice one almost always uses a combination of techniques