Image representation and compression via sparse solutions of systems of linear equations

Alfredo Nava-Tudela
John J. Benedetto, advisor
Multimedia flood
Cn u rd ths?

If you answered yes to the above question, then you have grasped what we are trying to do here, but for images. In the example above, we have compressed the sentence “can you read this?” to “cn u rd ths?,” which amounts to a reduction of six characters, 33% fewer characters than in the original sentence, but without compromising its meaning.
Problem

We can do something similar for images by way of the following algebraic trick. Suppose that you have the system of linear equations

\[
\begin{pmatrix}
A \\
\end{pmatrix}
\begin{pmatrix}
x \\
\end{pmatrix}
=
\begin{pmatrix}
b \\
\end{pmatrix}
\]

where \( \text{rank}(A) = n < m \).
Problem

It is a fact that there are an infinite number of solutions to equations of the type depicted below, provided $A$ is full-rank as in our case.

\[
\begin{pmatrix}
 n \\
 m
\end{pmatrix}
A
= 
\begin{pmatrix}
 x \\
 b
\end{pmatrix}
\]

This is what we can exploit to compress an image $I$. Suppose that we can somehow convert $I$ into a vector $b$ and that for some ad hoc matrix $A$ we can find a vector $x_0$ such that the number of non-zero entries of $x_0$, from now on written as $\|x_0\|_0$, is a lot smaller than the number of non-zero entries of vector $b$, $\|x_0\|_0 < \|b\|_0$ in our new notation. Then if we store or transmit $x_0$ instead of $b$ we would have compressed image $I$. 
Results

- Fine tuning and speedup of Orthogonal Matching Pursuit (OMP)
- Efficient QR implementation of OMP
- Comparison with SolveOMP, a publicly available OMP solution
- DCT+Haar compression vs DCT, Haar compressions
- PSNR, SSIM, and MSSIM error estimation and bit-rate vs distortion
- 1D vs 2D bases comparison
- Quantization and distortion estimate
“Sparsity” equals compression

We can then think of signal compression in terms of our problem

\[
\begin{bmatrix}
x \\
A \\
b
\end{bmatrix}
\begin{bmatrix}
n \\
m
\end{bmatrix}
= \begin{bmatrix}
x \\
\end{bmatrix}
\]

If \( x \) is sparse, \( b \) is dense, store \( x \)!
Definition of “sparse”

- The $l_0$ “norm”:
  
  $$||x||_0 = \# \{k : x_k \neq 0\}$$

- $(P_0)$: \(\min_x ||x||_0\) subject to \(||Ax - b||_2 = 0\)

- $(P_0^\varepsilon)$: \(\min_x ||x||_0\) subject to \(||Ax - b||_2 < \varepsilon\)

Observations: In practice, $(P_0^\varepsilon)$ is the working definition of sparsity as it is the only one that is computationally practical. Solving $(P_0^\varepsilon)$ is NP-hard [2].
Some theoretical results

**Definition:** The *spark* of a matrix $A$ is the minimum number of linearly dependent columns of $A$. We write $\text{spark}(A)$ to represent this number.

**Theorem:** If there is a solution $x$ to $Ax = b$, and $\|x\|_0 < \text{spark}(A) / 2$, then $x$ is the sparsest solution. That is, if $y \neq x$ also solves the equation, then $\|x\|_0 < \|y\|_0$.

**Observation:** Computing $\text{spark}(A)$ is combinatorial, therefore hard. Alternative?
Some theoretical results

Definition: The *mutual coherence* of a matrix $A$ is the number

$$\mu(A) = \max_{1 \leq k, j \leq m, \ k \neq j} \frac{|a_k^T a_j|}{\|a_k\|_2 \cdot \|a_j\|_2}.$$ 

Lemma: $\text{spark}(A) \geq 1 + 1/\mu(A)$.

Theorem: If $x$ solves $Ax = b$, and $\|x\|_0 < \frac{(1 + \mu(A)^{-1})}{2}$, then $x$ is the sparsest solution. That is, if $y \neq x$ also solves the equation, then $\|x\|_0 < \|y\|_0$.

Observation: $\mu(A)$ is a lot easier and faster to compute, but $1 + 1/\mu(A)$ far worse bound than $\text{spark}(A)$, in general.
Finding sparse solutions: OMP

Orthogonal Matching Pursuit algorithm:

Task: Approximate the solution of \((P_0) : \min_x \|x\|_0\) subject to \(Ax = b\).
Parameters: We are given the matrix \(A\), the vector \(b\), and the threshold \(\epsilon_0\).

Initialization: Initialize \(k = 0\), and set
- The initial solution \(x^0 = 0\).
- The initial residual \(r^0 = b - Ax^0 = b\).
- The initial solution support \(S^0 = \text{Support}\{x^0\} = \emptyset\).

Main Iteration: Increment \(k\) by 1 and perform the following steps:
- Sweep: Compute the errors \(\epsilon(j) = \min_{z_j} \|z_j a_j - r^{k-1}\|_2^2\) for all \(j\) using the optimal choice \(z_j^* = a_j^T r^{k-1} / \|a_j\|_2^2\).
- Update Support: Find a minimizer \(j_0\) of \(\epsilon(j): \forall j \notin S^{k-1}, \epsilon(j_0) \leq \epsilon(j)\), and update \(S^k = S^{k-1} \cup \{j_0\}\).
- Update Provisional Solution: Compute \(x^k\), the minimizer of \(\|Ax - b\|_2^2\) subject to \(\text{Support}\{x\} = S^k\).
- Update Residual: Compute \(r^k = b - Ax^k\).
- Stopping Rule: If \(\|r^k\|_2 < \epsilon_0\), stop. Otherwise, apply another iteration.

Output: The proposed solution is \(x^k\) obtained after \(k\) iterations.
Finding sparse solutions: OMP

Orthogonal Matching Pursuit algorithm:
Finding sparse solutions: OMP

Orthogonal Matching Pursuit algorithm:
Finding sparse solutions: OMP

Orthogonal Matching Pursuit algorithm:
One more theoretical result

**Theorem:** For a system of linear equations \( Ax = b \) (\( A \) an \( n \) by \( m \) matrix, \( n < m \), and \( \text{rank}(A) = n \)), if a solution \( x \) exists obeying \( \|x\|_0 < (1 + \mu(A)^{-1})/2 \), then an OMP run with threshold parameter \( \varepsilon_0 = 0 \) is guaranteed to find \( x \) exactly.
Implementation Fine Tuning

My initial OMP implementation wasn’t optimized for speed. I made some improvements:

The core of the algorithm is found in the following three steps. Modifying the approach to each of them cut execution times considerably.

- **Sweep:** Compute the errors $\epsilon(j) = \min_{z_j} ||z_j a_j - r^{k-1}||^2_2$ for all $j$ using the optimal choice $z_j^* = a_j^T r^{k-1} / ||a_j||^2_2$.
- **Update Support:** Find a minimizer $j_0$ of $\epsilon(j)$: $\forall j \notin S^{k-1}$, $\epsilon(j_0) \leq \epsilon(j)$, and update $S^k = S^{k-1} \cup \{j_0\}$.
- **Update Provisional Solution:** Compute $x^k$, the minimizer of $||Ax - b||^2_2$ subject to $Support\{x\} = S^k$. 
Implementation Fine Tuning,
Round 1: ompQRf

The first improvement came from computing $\|r_{k-1}\| \cos(\theta_j)$, where $\theta_j$ is the angle between $a_j$ and $r_{k-1}$. This number reflects how good an approximation to the residue $z_j a_j$ is, and it is faster to compute than $\varepsilon(j)$. We also kept track of the best approximant during the “Sweep” so that “Update Support” is done in a more efficient way compared to what we had done in ompQR.

Finally, we sweep only on the set of columns that have not been added to the support set, resulting in further time gains on the “Sweep” step when $k > 1$. 

4/17/12 Ph.D. Final Oral Exam 17
Implementation Fine Tuning, Round 2: ompQRf2

The “Update Provisional Solution” involves an $l_2$ minimization that corresponds to a least squares approximation. The preferred method of choice in this case is a QR decomposition of the restricted system.

We implemented this part of the algorithm by taking advantage of previous QR steps as opposed to compute each time a brand new QR decomposition of the updated matrix that resulted from increasing the support set $S^k$. 
Implementation Fine Tuning, Round 3: ompQRf3

Finally, we heed the advice of Matlab to allocate some variables for speed, this change saves time too:

Run times for 'experiment.m' (k = 2)

<table>
<thead>
<tr>
<th></th>
<th>Time (seconds)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ompQR</td>
<td>617.802467</td>
<td></td>
</tr>
<tr>
<td>ompQRf</td>
<td>360.192118, 1.715</td>
<td>speedup</td>
</tr>
<tr>
<td>ompQRf2</td>
<td>308.379138, 1.168</td>
<td>speedup</td>
</tr>
<tr>
<td>ompQRf3</td>
<td>298.622174, 1.032</td>
<td>speedup</td>
</tr>
</tbody>
</table>

Total speedup from ompQR to ompQRf3: **2.068**
(Matlab version 2010b)
Implementation and Validation

In light of the theoretical results, we can envision the following roadmap to validate an implementation of OMP.

- We have a simple theoretical criterion to guarantee both solution uniqueness and OMP convergence:

If $\mathbf{x}$ is a solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$, and $||\mathbf{x}||_0 < (1+\mu(\mathbf{A})^{-1})/2$, then $\mathbf{x}$ is the unique sparsest solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ and OMP will find it.

- Hence, given a full-rank $n$ by $m$ matrix $\mathbf{A}$ ($n < m$), compute $\mu(\mathbf{A})$, and find the largest integer $k$ smaller than or equal to $(1+\mu(\mathbf{A})^{-1})/2$. That is, $k = \text{floor}((1+\mu(\mathbf{A})^{-1})/2)$. 


Implementation and Validation

- Build a vector $\mathbf{x}$ with exactly $k$ non-zero entries and produce a right hand side vector $\mathbf{b} = \mathbf{A}\mathbf{x}$. This way, you have a known sparsest solution $\mathbf{x}$ to which to compare the output of any OMP implementation.

- Pass $\mathbf{A}$, $\mathbf{b}$, and $\varepsilon_0$ to OMP to produce a solution vector $\mathbf{x}_{\text{omp}} = \text{OMP}(\mathbf{A}, \mathbf{b}, \varepsilon_0)$.

- If OMP terminates after $k$ iterations and $\|\mathbf{A}\mathbf{x}_{\text{omp}} - \mathbf{b}\| < \varepsilon_0$, for all possible $\mathbf{x}$ and $\varepsilon_0 > 0$, then the OMP implementation would have been validated.

Caveat: The theoretical proofs assume infinite precision.
Validation Results

We ran two experiments:

1) \( A \in \mathbb{R}^{100 \times 200} \), with entries in \( N(0,1) \) i.i.d. for which \( \mu(A) = 0.3713 \), corresponding to \( k = 1 \leq K \).

2) \( A \in \mathbb{R}^{200 \times 400} \), with entries in \( N(0,1) \) i.i.d. for which \( \mu(A) = 0.3064 \), corresponding to \( k = 2 \leq K \).

Observations:
- \( A \) will be full-rank with probability 1.
- For full-rank matrices \( A \) of size \( n \times m \), the mutual coherence satisfies \( \mu(A) \geq \sqrt{(m - n)/(n \cdot (m - 1))} \). That is, the upper bound of \( K = (1 + \mu(A)^{-1})/2 \) can be made as big as needed, provided \( n \) and \( m \) are big enough.
Validation Results

For each matrix $A$, we chose 100 vectors with $k$ non-zero entries whose positions were chosen at random, and whose entries were in $\mathbb{N}(0,1)$.

Then, for each such vector $x$, we built a corresponding right hand side vector $b = Ax$.

Each of these vectors would then be the unique sparsest solution to $Ax = b$, and OMP should be able to find them.

Finally, given $\varepsilon_0 > 0$, if our implementation of OMP were correct, it should stop after $k$ steps (or less), and if $x_{\text{OMP}} = \text{OMP}(A, b, \varepsilon_0)$, then $\|b - A x_{\text{OMP}}\| < \varepsilon_0$. 
Validation Results

\( k = 1 \)
Validation Results

$k = 1$
Validation Results

$k = 1$
Validation Results

$k = 1$
Validation Results

$k = 1$
Validation Results

\( k = 2 \)
Validation Results

\[ k = 2 \]
Validation Results

$k = 2$
Validation Results

$k = 2$
Validation Results

$k = 2$

![Graph showing average number of iterations vs tolerance](image)
Reproducing Paper Results

For the first portion of our testing protocol, we set to reproduce the experiment described in section (3.3.1) of [1], limited to the results obtained for OMP.

\[ Ax = b, \] where \( A \) is 100 x 200, each column i.i.d. N(0,1), and \( x \) has \( k \) non-zero entries chosen at random and i.i.d. N(0,1).

Repeat 100 times, for each \( k = 1 \) to 70, the following experiment and count the number of successes:

With \( b \) having been set to \( Ax \), does \( x_{\text{omp}} = \text{omp}(A,b,1e-5) \) converge to \( x \) within the given tolerance?

Reproducing Paper Results

SolveOMP is SparseLab’s implementation of OMP (http://sparselab.stanford.edu/)

Cardinality of the solution

Probability of success

A is 100x200

4/17/12  Ph.D. Final Oral Exam  35
Image Compression: setup

We need a matrix $A$, and we consider the basis of Discrete Cosine Transform waveforms, and the basis generated by the Haar wavelet. JPEG, JPEG2000 inspired.
Image Compression: setup

We are going to partition an image in smaller square sub-matrices to be linearized.

How to linearize a matrix?
Image Compression: setup

Consider the matrix \( A = [\text{DCT}_1 \, \text{Haar}_1] \), where \( \text{DCT}_1 \) is the basis of 1-dimensional DCT waveforms, and \( \text{Haar}_1 \) is the basis of 1-dimensional Haar wavelet waveforms.
Consider the matrix $\mathbf{A} = [ \text{DCT}_{2,3} \; \text{Haar}_{2,3} ]$, where $\text{DCT}_{2,3}$ is the basis of 2-dimensional DCT waveforms, and $\text{Haar}_{2,3}$ is the basis of 2-dimensional Haar wavelet waveforms.
Image Compression: images

We selected 5 natural images to test the compression properties of $A$, and compare to compression via DCT or Haar alone, i.e. $B = [\text{DCT}]$, or $C = [\text{Haar}]$.
Normalized bit-rate

To study the tradeoff between error and compression, we need to introduce a measure of how many bits it takes to store our image. If our image $I$ is composed of $M$ sub-images $I_j$, and each can be represented by $x_j$, where $j = 1, \ldots, M$. Then the normalized bit-rate is

$$nbr(I, A, \varepsilon) = \sum ||x_j||_0/(n_1 \times n_2),$$

where each sub-image $I_j$ is of size $n_1 \times n_2$. 
Image Compression: Barbara
Image Compression: Boat

![Graph showing compression performance for different methods with varying tolerance.]
Image Compression: Elaine
Image Compression: Peppers
Image Compression: Stream
Error Estimation

Peak Signal-to-Noise Ratio (PSNR):

$$\text{PSNR} = 20 \log_{10}(\text{MAX}_X / \sqrt{\text{MSE}}), \text{ (units in dB)}$$

with $\text{MAX}_X = 255$, and $\text{MSE} = \sum_{i,j} (X(i,j) - Y(i,j))^2 / \text{nm}$.

Structural Similarity (SSIM), and Mean Structural Similarity (MSSIM) indices:

$$\text{SSIM}(x, y) = \frac{(2\mu_x \mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

$$\text{MSSIM}(X, Y) = \frac{1}{M} \sum_{j=1}^{M} \text{SSIM}(x_j, y_j)$$
Error Estimation: PSNR

Ideal error distribution. Consider an $L \times L$ image that has been linearized to a vector $\mathbf{b}$ of length $L^2$. Assume that the OMP approximation within $\varepsilon$ has distributed the error evenly, that is, if $\mathbf{y} = \mathbf{Ax}_{\text{omp}}$

$$\| \mathbf{Ax}_{\text{omp}} - \mathbf{b} \|_2 < \varepsilon \iff \| \mathbf{y} - \mathbf{b} \|_2^2 < \varepsilon^2$$

$$\iff \sum_{j=1,\ldots,L^2} (\mathbf{y}_j - \mathbf{b}_j)^2 < \varepsilon^2$$

$$\iff L^2 c^2 < \varepsilon^2$$

$$\iff c < \varepsilon/L$$

That is, if we want to be within $c$ units from each pixel, we have to choose a tolerance $\varepsilon$ such that $c$ is equal to $\varepsilon/L$. 
Image Compression: PSNR
Image Compression: PSNR
Image Compression: MSSIM
Image Compression: MSSIM

![Graph showing DCT+Haar normalized bit-rate, corresponding MSSIM](image-url)
Error Comparison

![Graph showing PSNR vs MSSIM for various images: Barbara, Boat, Elaine, Peppers, Stream. The x-axis represents MSSIM values ranging from 0 to 1, and the y-axis represents dB values ranging from 0 to 100. The graph shows the comparison between PSNR and MSSIM for different images, illustrating the trade-off between the two metrics.]
Error Comparison: Barbara
Error Comparison: Boat

Boat - OMP stoppage criteria (DCT+Haar)

- $\|\cdot\|_2$
- $\|\cdot\|_{\text{MSSIM}}$

$\text{bpp}$

$10^0$

$10^{-1}$

$10^{-2}$

$0.4$ $0.5$ $0.6$ $0.7$ $0.8$ $0.9$ $1$

MSSIM
Error Comparison: Elaine
Error Comparison: Peppers

Peppers - OMP stoppage criteria (DCT+Haar)
Error Comparison: Stream
Visual overview: Boat

\( \varepsilon = 200, \ c = 25 \)

PSNR = 25.2711 dB
MSSIM = 0.6006
n-bit-rate = 0.0217 bpp
Termination: \( \| \cdot \|_2 \)
Visual overview: Boat

\[ \varepsilon = 64, \ c = 8 \]
PSNR = 31.7332 dB
MSSIM = 0.8222
n-bit-rate = 0.0710 bpp
Termination: \[ \| \cdot \|_2 \]
Visual overview: Boat

\( \varepsilon = 32, c = 4 \)

PSNR = 36.6020 dB

MSSIM = 0.9214

n-bit-rate = 0.1608 bpp

Termination: \( \| \cdot \|_2 \)
Visual overview: Boat

$\delta = 0.92$
PSNR = 34.1405 dB
MSSIM = 0.9355
n-bit-rate = 0.1595 bpp
Termination: $\|\cdot\|_{\text{ssim}}$
Visual overview: Barbara

$\varepsilon = 32, \ c = 4$

PSNR = 36.9952 dB

MSSIM = 0.9447

n-bit-rate = 0.1863 bpp

Termination: $\| \cdot \|_2$
Visual overview: Barbara

\[ \delta = 0.94 \]
\[ \text{PSNR} = 32.1482 \text{ dB} \]
\[ \text{MSSIM} = 0.9466 \]
\[ \text{n-bit-rate} = 0.1539 \text{ bpp} \]

Termination: \[ \| . \|_{\text{ssim}} \]
1D vs 2D basis comparison
1D vs 2D basis comparison
1D vs 2D basis comparison
1D vs 2D basis comparison
1D vs 2D basis comparison
Quantization

\[
S_0 = (-\infty, a_0] \quad S_i = (a_{i-1}, a_i] \quad S_4 = (a_4, \infty)
\]

\[i = 1, 2, 3\]

\[S = \{S_i \subset \mathbb{R} : i \in I\} \quad C = \{y_i \in \mathbb{R} : i \in I\}\]

\[q(x) = \sum_{i \in I} y_i 1_{S_i}(x)\]
Quantization

\[ \alpha: \mathcal{A} \rightarrow I \quad \gamma: I \rightarrow J \]

\[ I = \{0, \ldots, m\} \]

\[ \gamma^{-1}: J \rightarrow I \]

\[ \beta: I \rightarrow \mathcal{A} \]

\[ \begin{array}{c}
  J \\
  00 \\
  01 \\
  11 \\
  011 \\
  010 \\
  \vdots
\end{array} \]
Quantization

Instantaneous rate

\[ r(x) = \frac{1}{k} \log(\gamma(\alpha(x))) \]

Distortion measure

\[ d(x, \hat{x}) = \|x - \hat{x}\|_2^2 \]

Normalized average rate

\[
R(\alpha, \gamma) = E[r(X)] = \frac{1}{k}E[\log(\gamma(\alpha(X)))]
= \frac{1}{k} \sum_i l(\gamma(i)) \int_{S_i} f(x) \, dx
\]

Normalized average distortion

\[
D(\alpha, \beta) = \frac{1}{k}E[d(X, \beta(\alpha(X)))]
= \frac{1}{k} \sum_i \int_{S_i} d(x, y_i) f(x) \, dx
\]
Quantization

Transform encoding

\[ x_0 = Tb = OMP(aA, b, \varepsilon_0) \quad \alpha = QT \quad d(\beta(\alpha(b)),b) = ? \]

\[ T' = aA \quad \beta = T'Q' \]

\[ b = aA \quad x_0 = T'Q'Qx_0 \]
\[ d(\beta(\alpha(b)), b) = \|T'Q'Q'Tb - b\|_2 \]

\[ = \|T'Q'Q'Tb - T'Tb + T'Tb - b\|_2 \]

\[ = \|aA\tilde{x}_0 - aAx_0 + aAx_0 - b\|_2 \]

\[ \leq \|aA\tilde{x}_0 - aAx_0\|_2 + \|aAx_0 - b\|_2 \]

\[ = ac\|\delta\|_{\infty}\|x_0\|_0 + \epsilon_0, \quad \delta = \tilde{x}_0 - x_0 \]
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
Histogram and coefficient distribution

2D basis elements with $c = 1/8$
PSNR vs bit-rate

Comparison between our work and published results
Future Work

- Fast algorithm for DCT+Haar?
- $\gamma$ functions for quantizer in the DCT+Haar setting
- Complete theory for Gabor systems and other matrices
Thank you!
Some References


https://ece.uwaterloo.ca/~z70wang/research/ssim/index.html