Sparse Recovery Using Sparse (Random) Matrices

Piotr Indyk
MIT

Joint work with: Radu Berinde, Anna Gilbert, Howard Karloff, Martin Strauss and Milan Ruzic

Linear Compression

(learning Fourier coeffs, linear sketching, finite rate of innovation, **compressed sensing...)**

- Setup:
  - Data/signal in n-dimensional space : x
    E.g., x is an 256x256 image $\Rightarrow$ n=65536
  - Goal: compress x into a "sketch" Ax ,
    where A is a m x n "sketch matrix", $m << n$
- Requirements:
  - Plan A: want to recover x from Ax
    - Impossible: undetermined system of equations
  - Plan B: want to recover an "approximation" x^* of x
    - Sparsity parameter k
    - Informally: want to recover largest $k<<n$ coordinates of x
    - Formally: want x^* such that

$$\|x^*-x\|_p \leq C(k)\ \min_{x'} \|x'-x\|_q$$

over all x' that are k-sparse (at most k non-zero entries)
- Want:
  - Good compression (small m)
  - Efficient algorithms for encoding and recovery
- Why linear compression ?

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

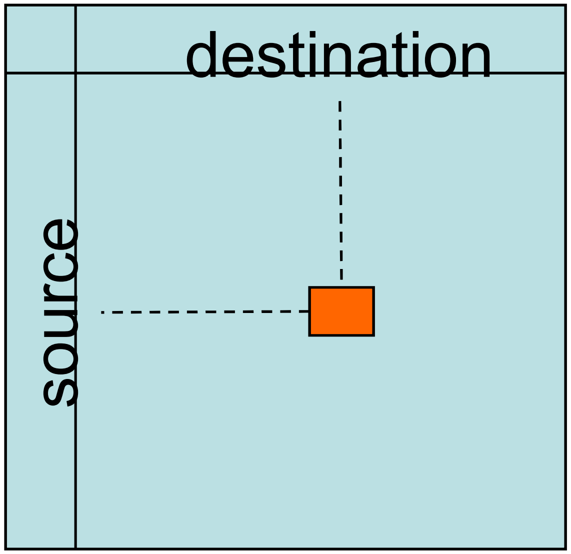
k=0.1n

$$\begin{pmatrix} & A & \end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} Ax \end{pmatrix}$$

Application I: Monitoring Network Traffic Data Streams

- Router routs packets
  - Where do they come from ?
  - Where do they go to ?

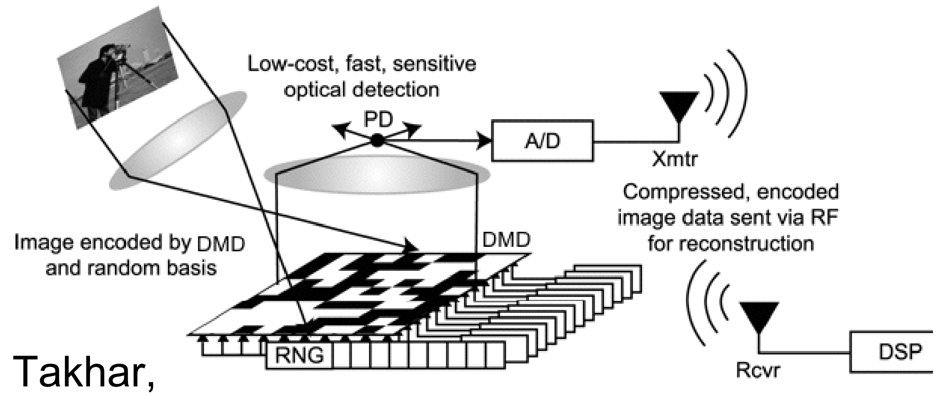
- Ideally, would like to maintain a traffic matrix $x[.,.]$
  - Easy to update: given a (src,dst) packet, increment $x_{src,dst}$
  - Requires way too much space! (2^{32} x 2^{32} entries)
  - Need to compress x, increment easily
- Using linear compression we can:
  - Maintain sketch Ax under increments to x, since
    $$A(x+\Delta) = Ax + A\Delta$$
  - Recover x^* from Ax



x

Applications, ctd.

- Single pixel camera



[Wakin, Laska, Duarte, Baron, Sarvotham, Takhar, Kelly, Baraniuk'06]

- Pooling Experiments
  [Kainkaryam, Woolf'08], [Hassibi et al'07], [Dai-Sheikh, Milenkovic, Baraniuk], [Shental-Amir-Zuk'09]

Constructing matrix A

- “Most” matrices A work
  - Sparse matrices:
    - Data stream algorithms
    - Coding theory (LDPCs)
  - Dense matrices:
    - Compressed sensing
    - Complexity/learning theory

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

- “Traditional” tradeoffs:
  - Sparse: computationally more efficient, explicit
  - Dense: shorter sketches
- Goal: the “best of both worlds”

Prior and New Results

Paper	Rand. / Det.	Sketch length	Encode time	Column sparsity	Recovery time	Approx

Scale: Excellent | Very Good | Good | Fair

Prior and New Results

Paper	R/D	Sketch length	Encode time	Column sparsity	Recovery time	Approx
[CCF'02], [CM'06]	R	k log n	n log n	log n	n log n	l2 / l2
	R	k $\log^c$ n	n $\log^c$ n	$\log^c$ n	k $\log^c$ n	l2 / l2
[CM'04]	R	k log n	n log n	log n	n log n	l1 / l1
	R	k $\log^c$ n	n $\log^c$ n	$\log^c$ n	k $\log^c$ n	l1 / l1
[CRT'04] [RV'05]	D	k log(n/k)	nk log(n/k)	k log(n/k)	n^c	l2 / l1
	D	k $\log^c$ n	n log n	k $\log^c$ n	n^c	l2 / l1
[GSTV'06] [GSTV'07]	D	k $\log^c$ n	n $\log^c$ n	$\log^c$ n	k $\log^c$ n	l1 / l1
	D	k $\log^c$ n	n $\log^c$ n	k $\log^c$ n	$k^2 \log^c$ n	l2 / l1
[BGIKS'08]	D	k log(n/k)	n log(n/k)	log(n/k)	n^c	l1 / l1
[GLR'08]	D	k $\log n^{\log\log\log n}$	kn^{1-a}	n^{1-a}	n^c	l2 / l1
[NV'07], [DM'08], [NT'08], [BD'08], [GK'09], …	D	k log(n/k)	nk log(n/k)	k log(n/k)	nk log(n/k) * log	l2 / l1
	D	k $\log^c$ n	n log n	k $\log^c$ n	n log n * log	l2 / l1
[IR'08], [BIR'08],[BI'09]	D	k log(n/k)	n log(n/k)	log(n/k)	n log(n/k)* log	l1 / l1

"state of art"

Recovery “in principle”
(when is a matrix “good”)

dense vs. sparse

- Restricted Isometry Property (RIP) * - sufficient property of a dense matrix A:

$$\Delta \text{ is } k\text{-sparse} \Rightarrow ||\Delta||_2 \le ||A\Delta||_2 \le C\, ||\Delta||_2$$

- Holds w.h.p. for:
  - Random Gaussian/Bernoulli: $m=O(k \log (n/k))$
  - Random Fourier: $m=O(k \log^{O(1)} n)$
- Consider $m \times n$ 0-1 matrices with d ones per column
- Do they satisfy RIP ?
  - No, unless $m=\Omega(k^2)$ [Chandar'07]
- However, they can satisfy the following RIP-1 property [Berinde-Gilbert-Indyk-Karloff-Strauss'08]:

$$\Delta \text{ is } k\text{-sparse} \Rightarrow d\,(1-\varepsilon)\, ||\Delta||_1 \le ||A\Delta||_1 \le d||\Delta||_1$$

- Sufficient (and necessary) condition: the underlying graph is a $(\, k,\ d(1-\varepsilon/2)\,)$-expander

Expanders

n

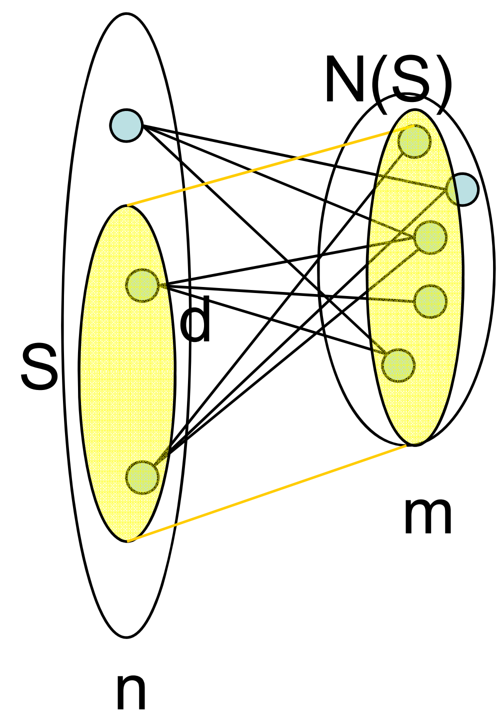
QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

m

- A bipartite graph is a $(k,d(1-\varepsilon))$-expander if for any left set S, $|S| \leq k$, we have $|N(S)| \geq (1-\varepsilon) d\,|S|$
- Objects well-studied in theoretical computer science and coding theory
- Constructions:
  - Probabilistic: $m=O(k \log (n/k))$
  - Explicit: $m=k$ quasipolylog n
- High expansion implies RIP-1:

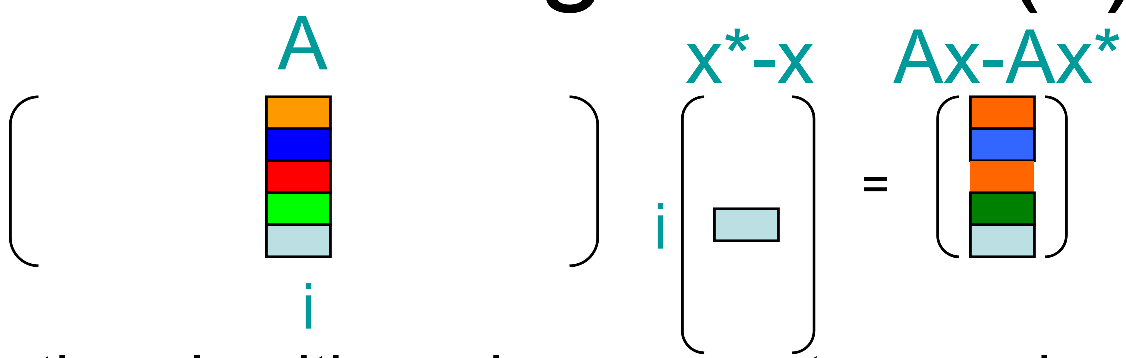
  Δ is k-sparse $\Rightarrow d\,(1-\varepsilon)\,||\Delta||_1 \leq ||A\Delta||_1 \leq d||\Delta||_1$

  [Berinde-Gilbert-Indyk-Karloff-Strauss'08]



Recovery: algorithms

Matching Pursuit(s)



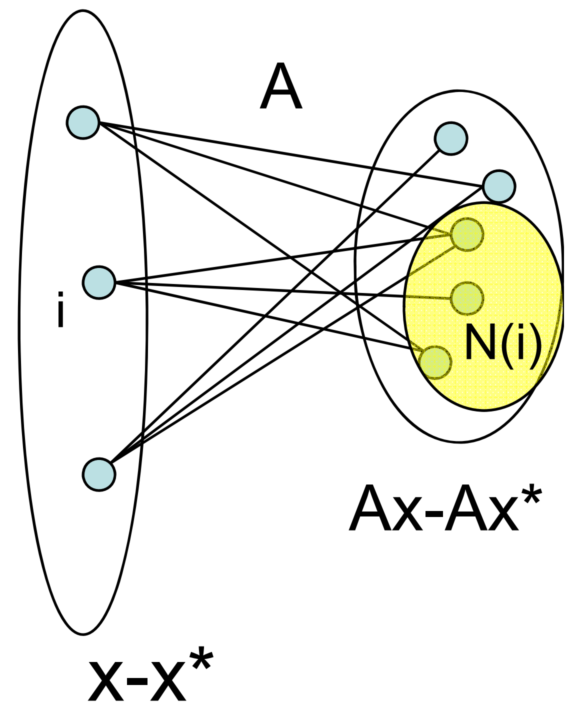
- Iterative algorithm: given current approximation x^* :
  – Find (possibly several) i s. t. A_i “correlates” with $Ax\text{-}Ax^*$. This yields i and z s. t.

$$||x^*+ze_i-x||_p << ||x^* - x||_p$$

  – Update x^*
  – Sparsify x^* (keep only k largest entries)
  – Repeat
- Norms:
  – $p=2$: CoSaMP, SP, IHT etc (RIP)
  – $p=1$: SMP, SSMP (RIP-1)
  – $p=0$: LDPC bit flipping (sparse matrices)

Sequential Sparse Matching Pursuit

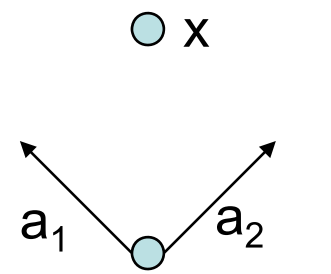
- Algorithm:
  - $x^*=0$
  - Repeat T times
    - Repeat $S=O(k)$ times
      - Find i and z that minimize* $\|A(x^*+ze_i)-Ax\|_1$
      - $x^* = x^*+ze_i$
    - Sparsify x^*
      (set all but k largest entries of x^* to 0)
- Similar to SMP, but updates done sequentially

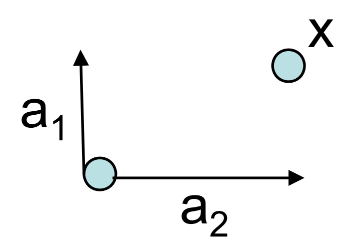


* Set $z=\text{median}[\,(Ax^*-Ax)_{N(i)}$

SSMP: Approximation guarantee

- Want to find k-sparse x^* that minimizes $||x-x^*||_1$
- By RIP1, this is approximately the same as minimizing $||Ax-Ax^*||_1$
- Need to show we can do it *greedily*

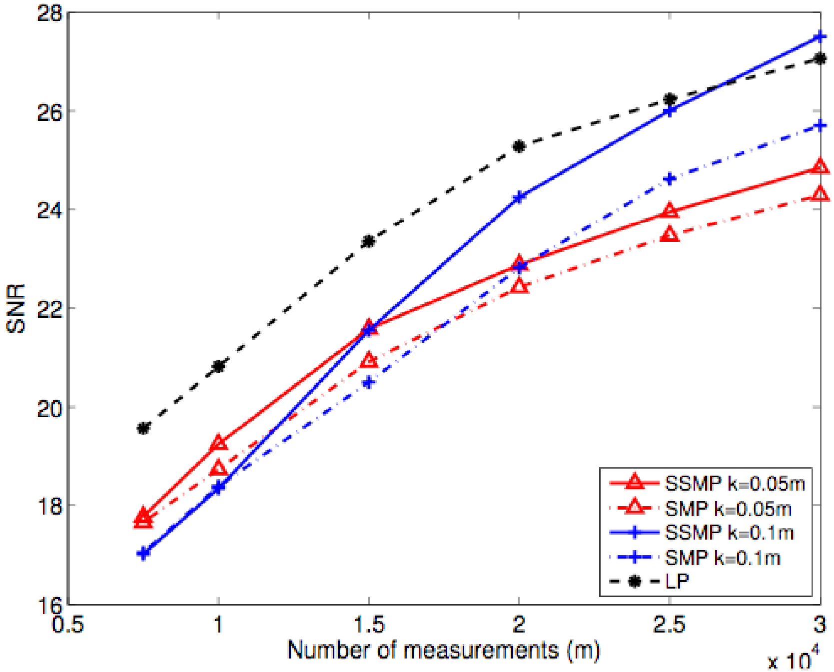


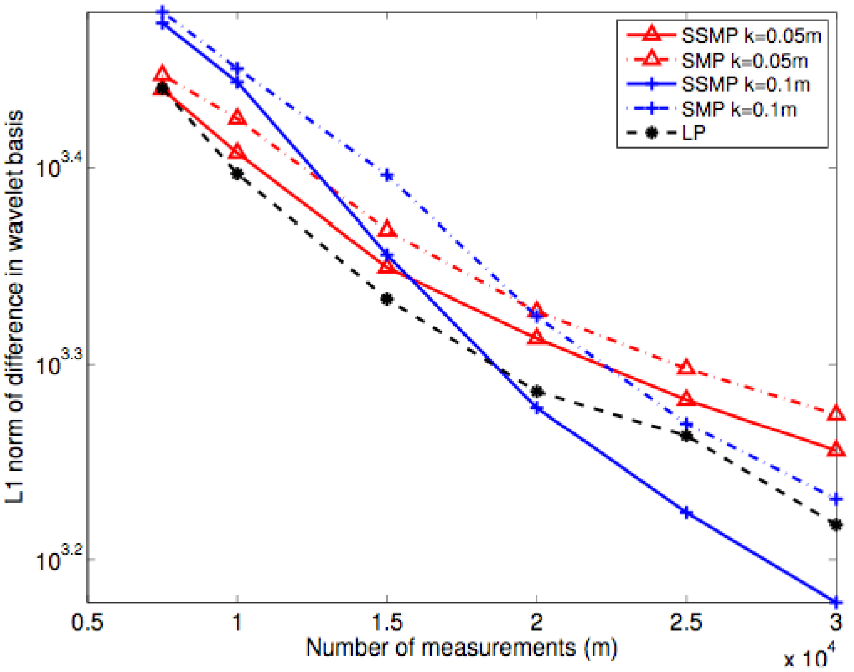


Supports of a_1 and a_2 have small overlap (typically)

Experiments

256x256





SSMP is ran with S=10000,T=20. SMP is ran for 100 iterations. Matrix sparsity is d=8.

Conclusions

- Algorithms for sparse approximation using sparse matrices
  - Fast
  - Short sketches
- State of the art: can do 2 out of 3:
  - Near-linear encoding/decoding
  - O(k log (n/k)) sketch length
  - Approximation guarantee with respect to L2/L1 norm
- Questions:
  - 3 out of 3 ?
  - Explicit constructions

This talk

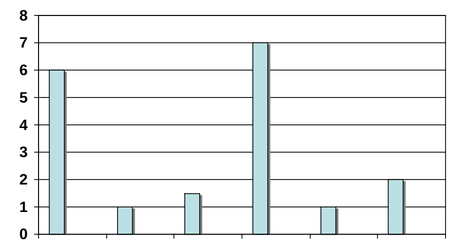
THANKS!

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

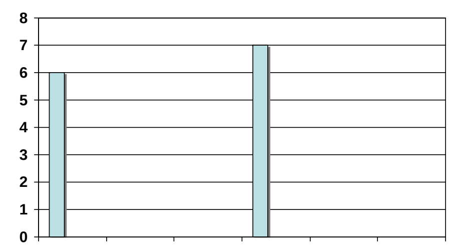
QuickTime™ and
TIFF (LZW) decompr
are needed to see this p

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

x



x*



k=2

Goal of this talk

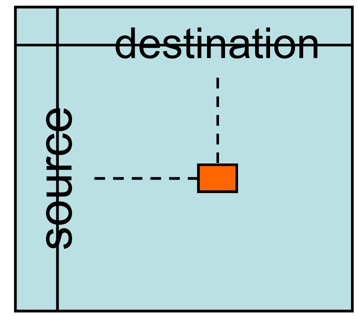
“Stay awake until the end”

Linear compression: applications

- Data stream algorithms

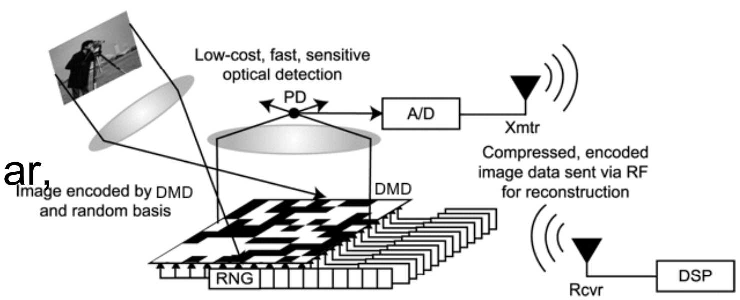
  (e.g. for network monitoring)
  - Efficient increments:

    $$A(x+\Delta) = Ax + A\Delta$$



- Single pixel camera

  [Wakin, Laska, Duarte, Baron, Sarvotham, Takhar, Kelly, Baraniuk'06]



- Pooling, Microarray Experiments [Kainkaryam, Woolf], [Hassibi et al], [Dai-Sheikh, Milenkovic, Baraniuk]

SSMP: Running time

- Algorithm:
  - $x^*=0$
  - Repeat T times
    - For each $i=1\ldots n$ compute* z_i that achieves

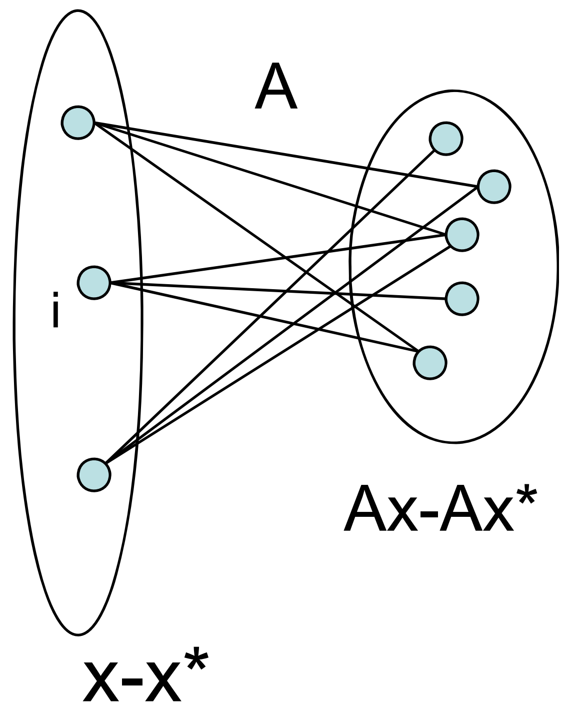
      $$D_i=\min_z \|A(x^*+ze_i)-b\|_1$$

      and store D_i in a heap
    - Repeat $S=O(k)$ times
      - Pick i,z that yield the best gain
      - Update $x^* = x^*+ze_i$
      - Recompute and store $D_{i'}$ for all i' such that $N(i)$ and $N(i')$ intersect
    - Sparsify x^*

      (set all but k largest entries of x^* to 0)
- Running time:

$$T\ [\ n(d+\log n) + k\ nd/m^*d\ (d+\log n)]$$

$$= T\ [\ n(d+\log n) + nd\ (d+\log n)] = T\ [\ nd\ (d+\log n)]$$



Proof: $d(1-\varepsilon/2)$-expansion $\Rightarrow$ RIP-1

- Want to show that for any k-sparse Δ we have

  $$d(1-\varepsilon)\,||\Delta||_1 \le ||A\Delta||_1 \le d||\Delta||_1$$

- RHS inequality holds for ***any*** Δ
- LHS inequality:
  - W.l.o.g. assume

    $$|\Delta_1| \ge \ldots \ge |\Delta_k| \ge |\Delta_{k+1}| = \ldots = |\Delta_n| = 0$$

  - Consider the edges $e=(i,j)$ in a lexicographic order
  - For each edge $e=(i,j)$ define $r(e)$ s.t.
    - $r(e)=-1$ if there exists an edge $(i',j)<(i,j)$
    - $r(e)=1$ if there is no such edge
- Claim 1: $||A\Delta||_1 \ge \sum_{e=(i,j)} |\Delta_i| r_e$
- Claim 2: $\sum_{e=(i,j)} |\Delta_i| r_e \ge (1-\varepsilon)\, d||\Delta||_1$