Sparse Matrices in Streaming and Sparse Recovery

Piotr Indyk

MIT

Basic Data Stream Model

- Single pass over the data: $i_1, i_2, \ldots, i_n$
  - Typically, we assume n is known
- Bounded storage (typically n^{α} or $\log^c n$)
  - Units of storage: bits, words, coordinates or „elements” (e.g., points, nodes/edges)
- Fast processing time per element
- Randomness OK (in fact, almost always necessary)

8 2 1 9 1 9 2 4 6 3 9 4 2 3 4 2 3 8 5 2 5 6 ...

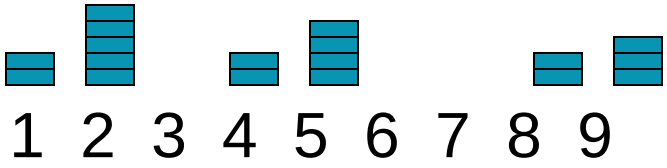
Counting Distinct Elements

- Stream elements: numbers from $\{1...m\}$
- Goal: estimate the number of distinct elements DE in the stream
  - Up to $1\pm\varepsilon$
  - With probability 1-P
- Simpler goal: for a given $T>0$, provide an algorithm which, with probability 1-P:
  - Answers YES, if $DE> (1+\varepsilon)T$
  - Answers NO, if $DE< (1-\varepsilon)T$

Vector Interpretation

Stream: 8 2 1 9 1 9 2 4 4 9 4 2 5 4 2 5 8 5 2 5

Vector X:



- Initially, $x=0$
- Insertion of i is interpreted as

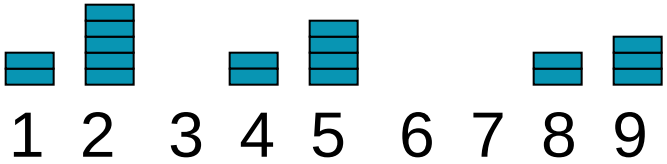
$$x_i = x_i + 1$$

- Want to estimate the sparsity of x (number of non-zero elements)

Estimating DE(x)

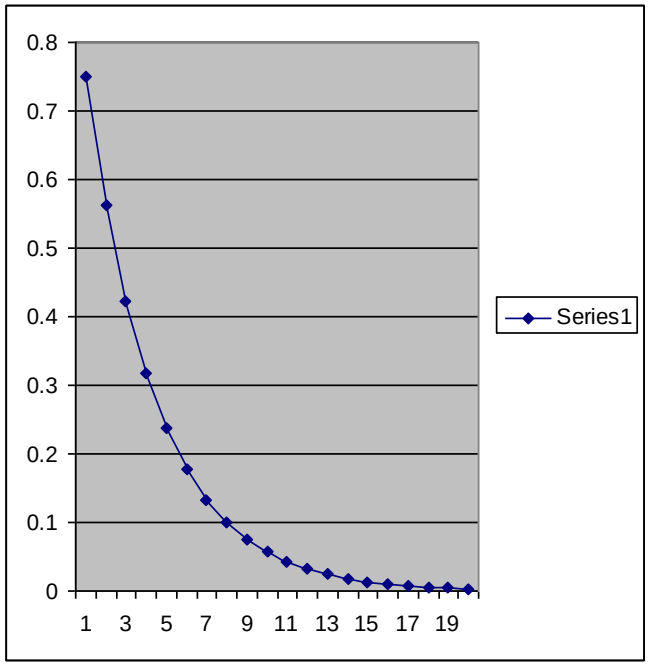
(variant of) [Flajolet-Martin'85]

Vector X:



Set S: + + + (T=4)

- Choose a random set S of coordinates
  - For each i, we have Pr[i∈S]=1/T
  - Implemented by choosing a hash function h: {1..m} -> {1..T} and setting S={i: h(i)=1}
- Maintain $Sum_S(x) = \Sigma_{i \in S} x_i$
- Estimation algorithm I:
  - YES, if $Sum_S(x)>0$
  - NO, if $Sum_S(x)=0$
- Analysis:
  - $Pr=Pr[Sum_S(x)=0] = (1-1/T)^{DE}$
  - For T large enough: $(1-1/T)^{DE} \approx e^{-DE/T}$
  - Using calculus, for ε small enough:
    - If $DE> (1+\varepsilon)T$, then $Pr \approx e^{-(1+\varepsilon)} < 1/e - \varepsilon/3$
    - if $DE< (1-\varepsilon)T$, then $Pr \approx e^{-(1-\varepsilon)} > 1/e + \varepsilon/3$



Estimating DE(x) ctd.

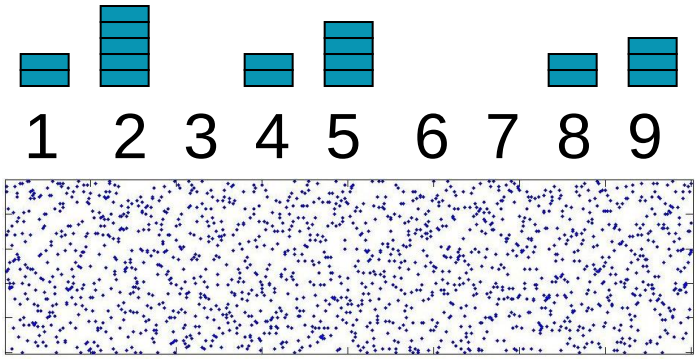
- We have Algorithm I:
  - If DE> $(1+\varepsilon)T$, then Pr<$1/e-\varepsilon/3$
  - if DE< $(1-\varepsilon)T$, then Pr>$1/e+\varepsilon/3$
- Algorithm II:
  - Select sets $S_1 \ldots S_k$, $k=O(\log(1/P)/\varepsilon^2)$
  - Let Z = number of $Sum_{Sj}(x)$ that are equal to 0
  - By Chernoff bound, with probability >1-P
    - If DE> $(1+\varepsilon)T$, then Z<k/e
    - if DE< $(1-\varepsilon)T$, then Z>k/e

- Total space (decision version): $O(\log (1/P)/\varepsilon^2)$ numbers in range 0…n

- Better algorithms known:
  - Theory: $O(1/\cdot^2 +\log n)$ bits [Kane-Nelson-Woodruff'10]
  - Practice: need 128 bytes for all works of Shakespeare , $\varepsilon\approx 10\%$ [Durand-Flajolet'03]

Sparse matrices

Vector X:



- The algorithm uses "linear sketches"

  $Sum_{Sj}(x)=\Sigma_{i\in Sj}\ x_i$
- I.e., the algorithm maintains a vector Ax where A is a sparse matrix (for large T)
- Can implement decrements $x_i=x_i-1$
  - I.e., the stream can contain deletions of elements (as long as $x\geq 0$)
  - Other names: dynamic model, turnstile model

Sparse Recovery

(approximation theory, statistical model selection, 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: underdetermined system of equations
  - Plan B: want to recover an "approximation" x* of x
    - Sparsity parameter k
    - Informally: want to recover largest k coordinates of x
    - Formally: want x* such that

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

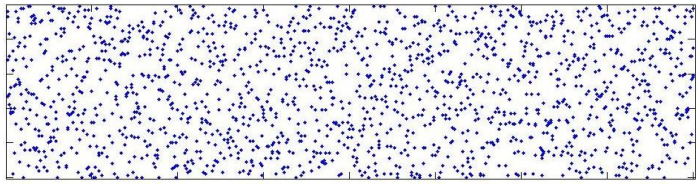
      over all x' that are k-sparse (at most k non-zero entries)

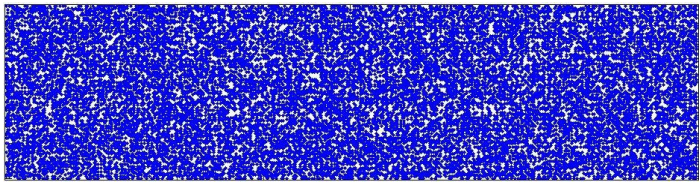
$$\begin{pmatrix} & A & \end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} Ax \end{pmatrix}$$

- Want:
  - Good compression (small m=m(k,n))
  - Efficient algorithms for encoding and recovery
- Why linear compression ?
  - Greater functionality!
  - Useful in data stream algorithms, compressed sensing, group testing, etc

Constructing matrix A

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





- "Traditional" tradeoffs:
  - Sparse: computationally more efficient, explicit
  - Dense: shorter sketches
- Recent results: the "best of both worlds"

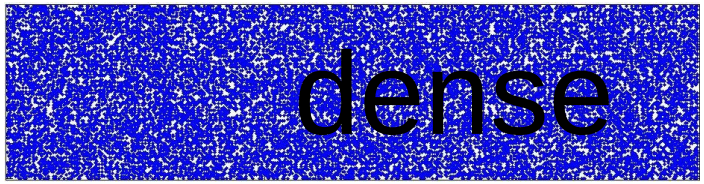
(using sparse matrices)

Scale: Excellent | Very Good | Good | Fair

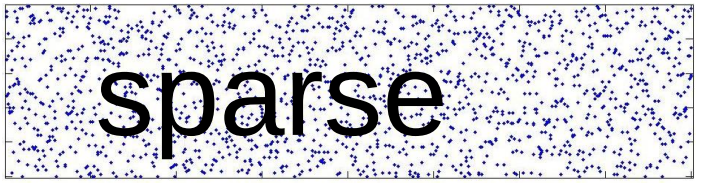
Results

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)	l1 / l1
[GLSP'10]	R	k log(n/k)	n $\log^c$ n	$\log^c$ n	k $\log^c$ n	l2 / l1

Caveats: (1) most "dominated" results not shown (2) only results for general vectors x are displayed
(3) sometimes the matrix type matters (Fourier, etc)



vs.



- Restricted Isometry Property (RIP) [Candes-Tao'04]

  Δ is k-sparse $\Rightarrow ||\Delta||_2 \leq ||A\Delta||_2 \leq C\,||\Delta||_2$

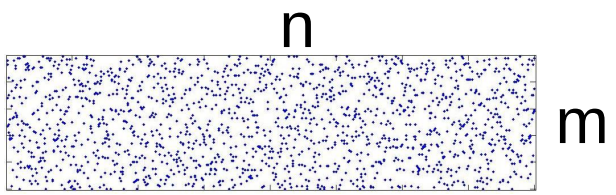
- Holds w.h.p. for the following ensembles of dense matrices:
  - Random Gaussian/Bernoulli: $m=O(k \log (n/k))$
  - Random Fourier: $m=O(k \log^{O(1)} n)$
- Consider $m \times n$ 0-1 matrices with d ones per column
- Do they satisfy RIP ?
  - No, unless $m=\Omega(k^2)$ [Chandar'07]
- However, they can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

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

- Sufficient (and necessary) condition: the underlying graph is a

  $(k, d(1-\varepsilon/2))$-expander

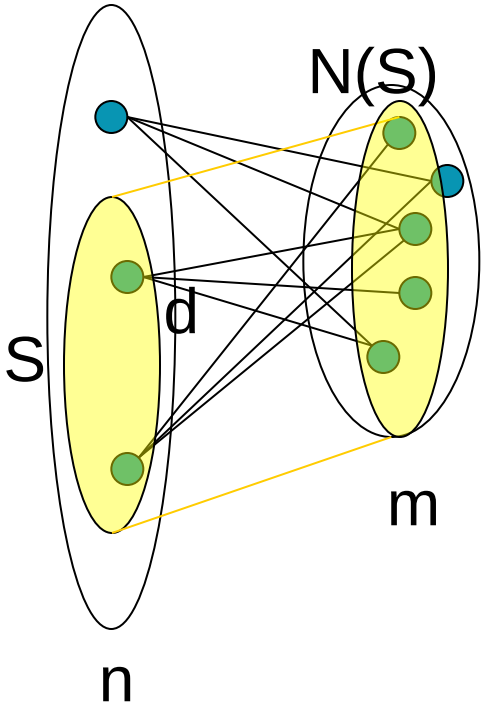
Expanders



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

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

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



Conclusions

- Sparse matrices – a powerful tool for streaming algorithms and sparse recovery
  - Distinct element estimation
  - Sparse recovery
- Open problems (next slide)

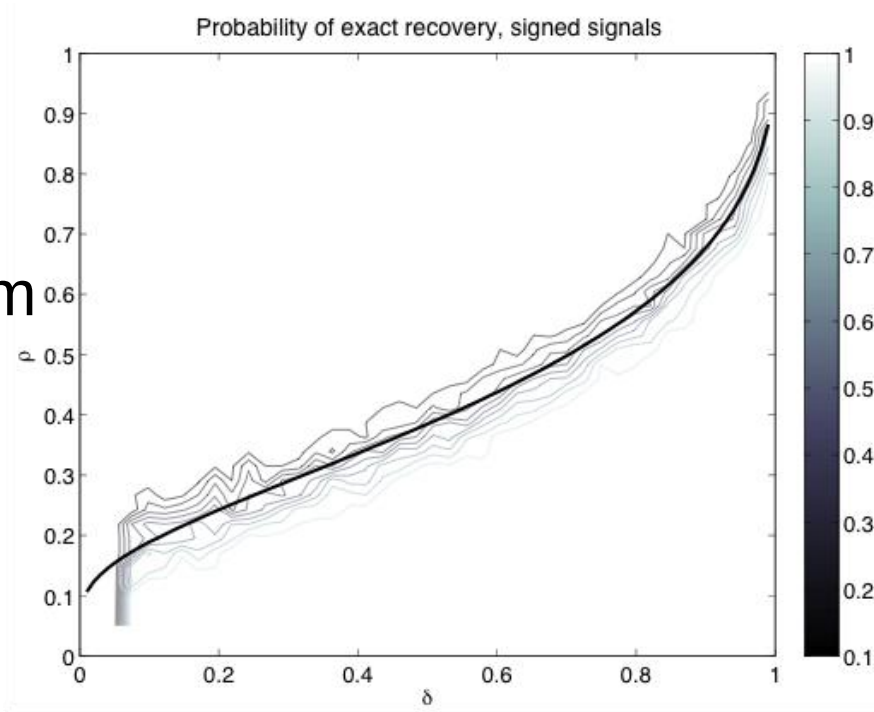
Open problem

- Sparse recovery algorithm:

  minimize $||x^*||_1$

  subject to $Ax=Ax^*$

- Theoretical analysis: the sufficient number of measurements is asymptotically the same for Gaussian and random sparse matrices, i.e. O(k log(n/k))
- Experimental performance
  - Thick line: probability =1/2 for Gaussian matrices
  - Thin lines: empirical probability levels for sparse matrices (d=10)
  - Same performance…
  - Prove this!



Part I

Paper	R/ D	Sketch length	Encode time	Column sparsity	Recovery time	Approx
[CM'04]	R	k log n	n log n	log n	n log n	l1 / l1

Theorem: There is a distribution over mxn matrices A, m=O(k logn), such that for any x, given Ax, we can recover x* such that

$$||x-x^*||_1 \leq C\, Err_1 \text{ , where } Err_1 = \min_{k\text{-sparse } x'} ||x-x'||_1$$

with probability 1-1/n.

The recovery algorithm runs in O(n log n) time.

This talk:

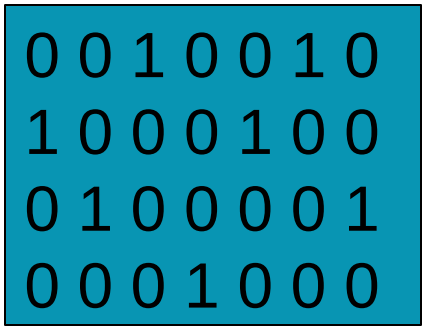
- Assume $x \geq 0$ – this simplifies the algorithm and analysis; see the original paper for the general case
- Prove the following l_∞/l_1 guarantee: $||x-x^*||_\infty \leq C\, Err_1 / k$

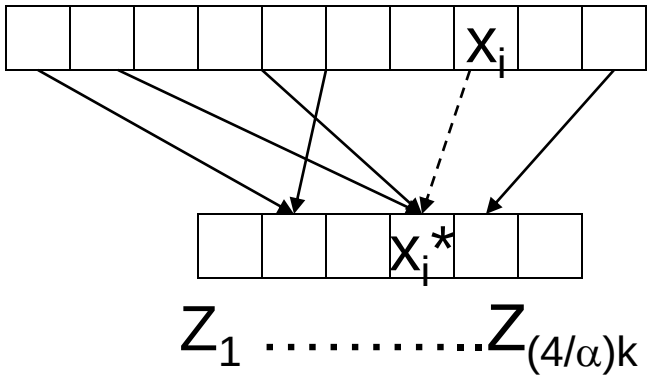
  This is actually stronger than the l_1/l_1 guarantee (cf. [CM'06], see also the Appendix)

Note: [CM'04] originally proved a weaker statement where $||x-x^*||_\infty \leq C||x||_1 / k$. The stronger guarantee follows from the analysis of [CCF'02] (cf. [GGIKMS'02]) who applied it to Err_2

First attempt

- Matrix view:
  - A 0-1 wxn matrix A, with one 1 per column
  - The i-th column has 1 at position h(i), where h(i) be chosen uniformly at random from {1…w}
- Hashing view:
  - Z=Ax
  - h hashes coordinates into "buckets" $Z_1 \ldots Z_w$
- Estimator: $x_i^* = Z_{h(i)}$

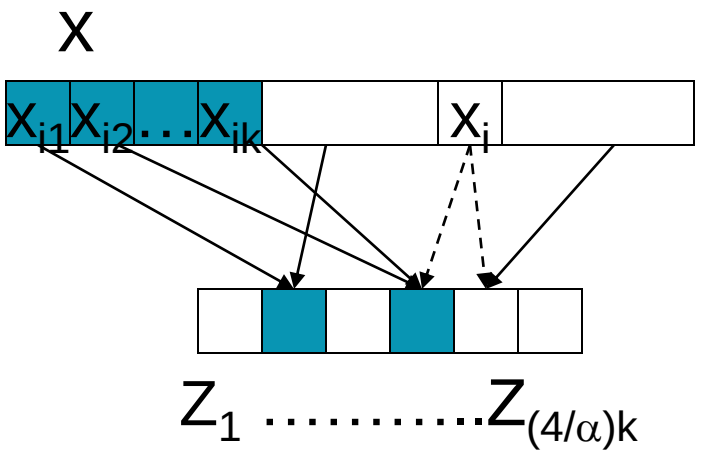




Closely related: [Estan-Varghese'02], "counting"/"spectral" Bloom filters

Analysis

- We show
  $x_i^* \leq x_i \pm \alpha\ Err/k$
  with probability >1/2
- Assume
  $|x_{i1}| \geq \ldots \geq |x_{im}|$
  and let S={i1…ik} (“elephants”)
- When is $x_i^* > x_i \pm \alpha\ Err/k$?



  - **Event 1**: S and i collide, i.e., h(i) in h(S-{i})
    Probability: at most $k/(4/\alpha)k = \alpha/4<1/4$ (if $\alpha<1$)
  - **Event 2**: many “mice” collide with i., i.e.,
    $\sum_{t \text{ not in } S \cup \{i\},\ h(i)=h(t)} x_t > \alpha\ Err/k$
    Probability: at most ¼ (Markov inequality)

- Total probability of “bad” events <1/2

Second try

- Algorithm:
  - Maintain d functions $h_1 \ldots h_d$ and vectors $Z^1 \ldots Z^d$
  - Estimator:

$$X_i^* = \min_t Z^t_{ht(i)}$$

- Analysis:
  - $\Pr[|x_i^* - x_i| \geq \alpha \, Err/k] \leq 1/2^d$
  - Setting $d=O(\log n)$ (and thus $m=O(k \log n)$) ensures that w.h.p

$$|x^*_i - x_i| < \alpha \, Err/k$$

Part II

Paper	R/ D	Sketch length	Encode time	Column sparsity	Recovery time	Approx
[BGIKS'08]	D	k log(n/k)	n log(n/k)	log(n/k)	n^c	l1 / 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

Proof: d(1-ε/2)-expansion ⇒ RIP-1

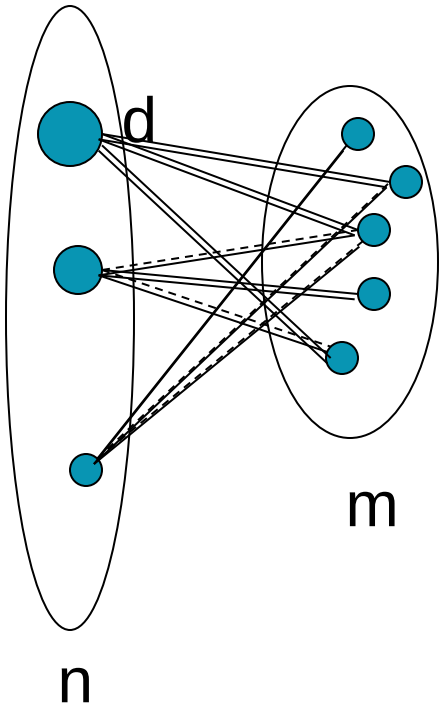
- Want to show that for any k-sparse Δ we have

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

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

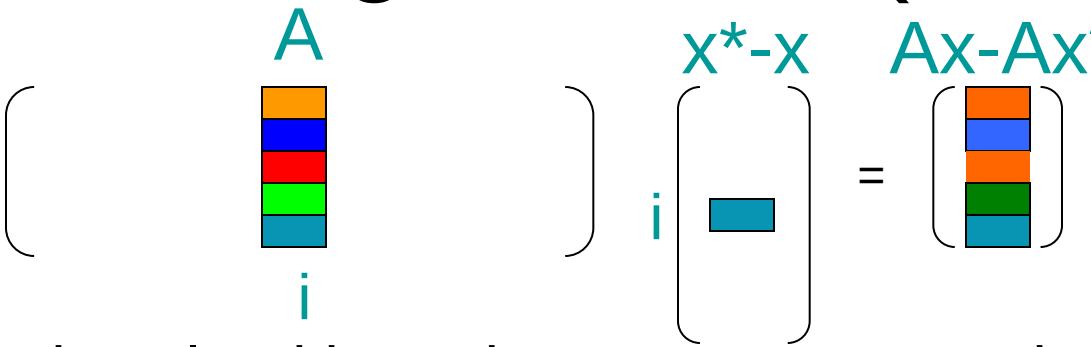
    $$|\Delta_1| \ge \ldots \ge |\Delta_k| \ge |\Delta_{k+1}| = \ldots = |\Delta_n| = 0$$

  - 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: $\sum_{e=(i,j)} |\Delta_i| r_e \ge (1-\varepsilon)\, d||\Delta||_1$



Recovery: algorithms

Matching Pursuit (xMP™)



- Iterative algorithm: given current approximation x* :
  - Find (possibly several) i s. t. A_i "correlates" with Ax-Ax* . This yields i and z s. t.

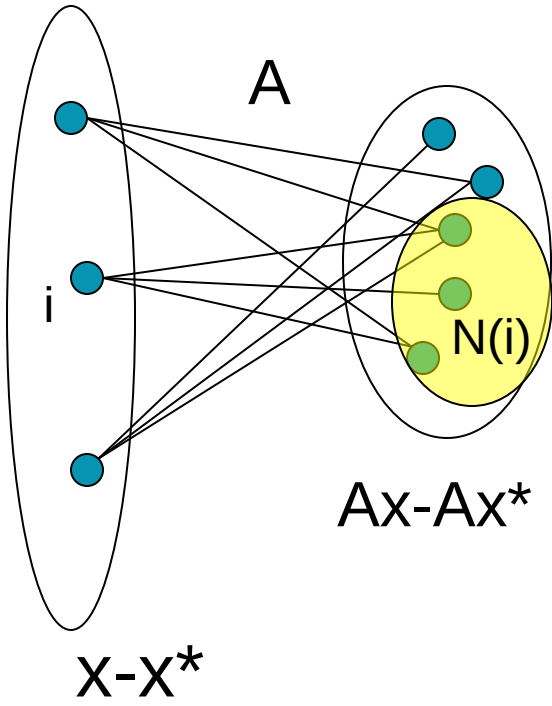
$$||x^*+ze_i-x||_p << ||x^* - x||_p$$

  - 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 [Berinde-Indyk'09]

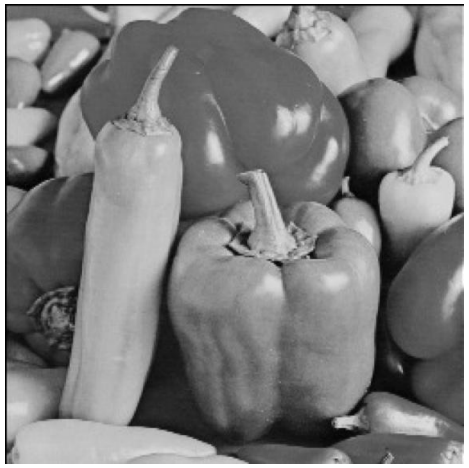
- Algorithm:
  - $x^*=0$
  - Repeat T times
    - Repeat $S=O(k)$ times
      - Find i and z that minimize* $||A(x^*+ze_i)-Ax||_1$
      - $x^* = x^*+ze_i$
    - Sparsify x^*

      (set all but k largest entries of x^* to 0)
- Similar to SMP, but updates done sequentially

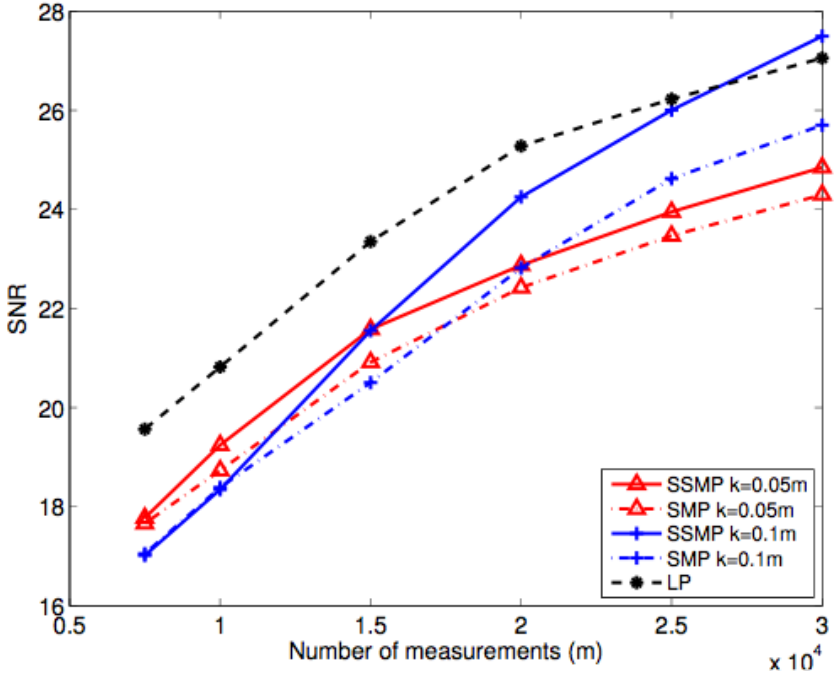


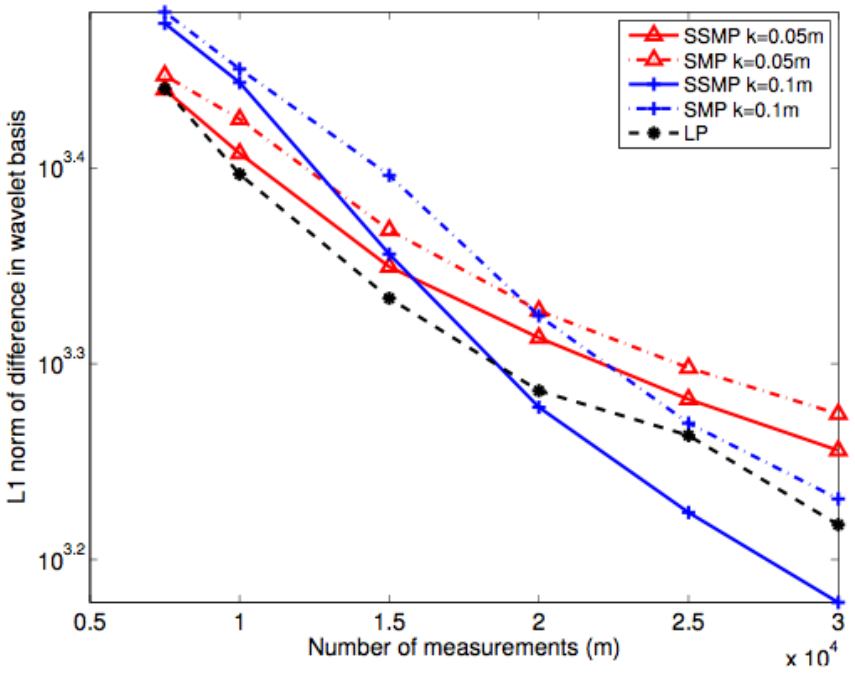
* Set $z=\text{median}[\ (Ax^*-Ax)_{N(I)}$.Instead, one could first optimize (gradient) i and then z [Fuchs'09]

Experiments



256x256





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

Conclusions

- Sparse approximation using sparse matrices
- State of the art: deterministically can do 2 out of 3:
  - Near-linear encoding/decoding
  - O(k log (n/k)) measurements
  - Approximation guarantee with respect to L2/L1 norm

  } This talk
- Open problems:
  - 3 out of 3 ?
  - O(k ~~log (n/k)~~) measurements by restricting sparsity patterns ?

References

- Survey:

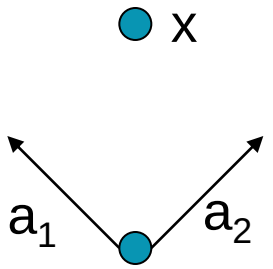
  A. Gilbert, P. Indyk, “Sparse recovery using sparse matrices”, Proceedings of IEEE, June 2010.

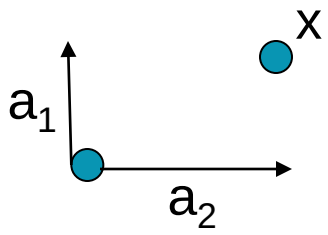
- Courses:
  - “Streaming, sketching, and sub-linear space algorithms”, Fall’07
  - “Sub-linear algorithms” (with Ronitt Rubinfeld), Fall’10
- Blogs:
  - Nuit blanche: nuit-blanche.blogspot.com/

Appendix

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)

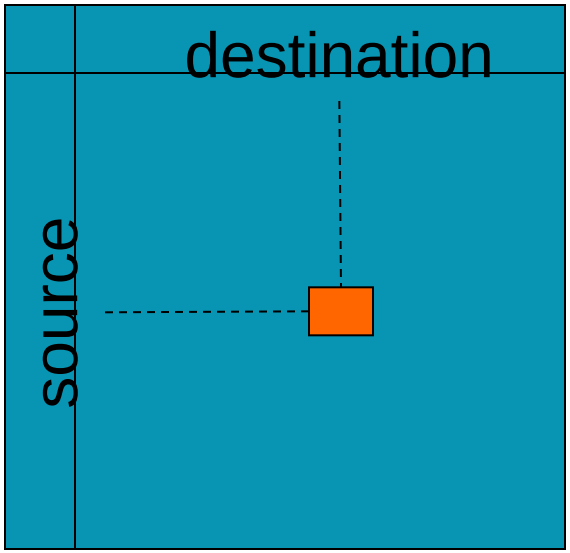
l_∞/l_1 implies l_1/l_1

- Algorithm:
  - Assume we have x^* s.t. $||x-x^*||_\infty \leq C\ Err_1/k$.
  - Let vector x' consist of k largest (in magnitude) elements of x^*
- Analysis
  - Let S (or S^*) be the set of k largest in magnitude coordinates of x (or x^*)
  - Note that $||x^*_S|| \leq ||x^*_{S^*}||_1$
  - We have

$$
\begin{aligned}
||x-x'||_1 &\leq ||x||_1 - ||x_{S^*}||_1 + ||x_{S^*}-x^*_{S^*}||_1 \\
&\leq ||x||_1 - ||x^*_{S^*}||_1 + 2||x_{S^*}-x^*_{S^*}||_1 \\
&\leq ||x||_1 - ||x^*_S||_1 + 2||x_{S^*}-x^*_{S^*}||_1 \\
&\leq ||x||_1 - ||x_S||_1 + ||x^*_S-x_S||_1 + 2||x_{S^*}-x^*_{S^*}||_1 \\
&\leq Err + 3\alpha/k * k \\
&\leq (1+3\alpha)Err
\end{aligned}
$$

Application I: Monitoring Network Traffic Data Streams

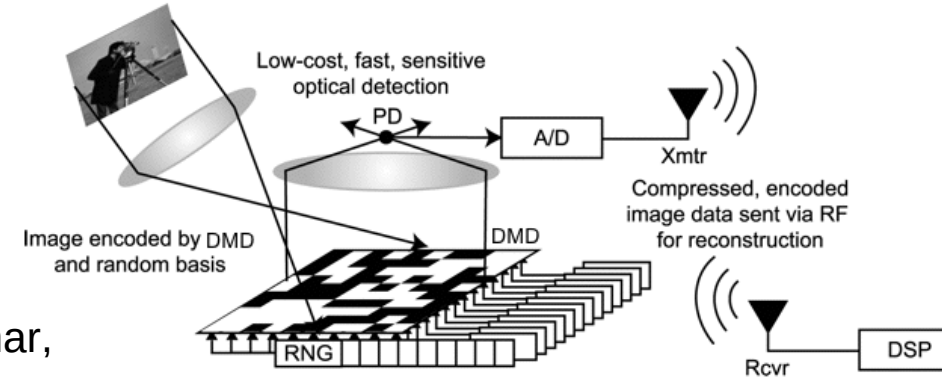
- Router routs packets
  - Where do they come from ?
  - Where do they go to ?
- Ideally, would like to maintain a traffic matrix x[.,.]
  - Easy to update: given a (src,dst) packet, increment $x_{src,dst}$
  - Requires way too much space! (2^{32} x 2^{32} entries)
  - Need to compress x, increment easily
- 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.

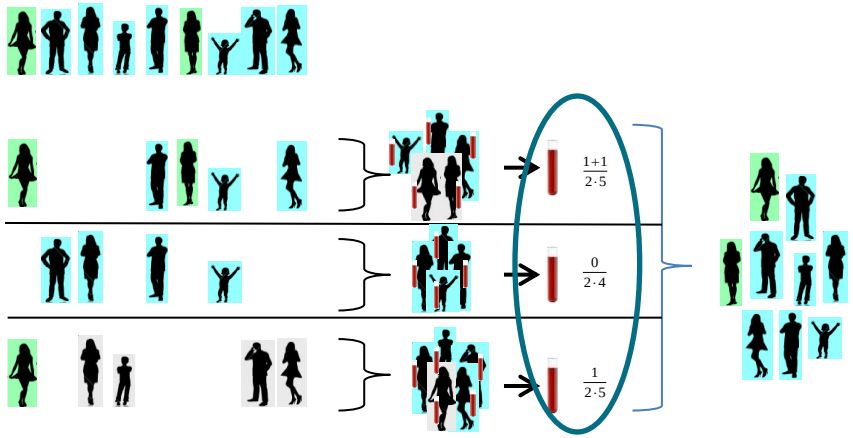
- Single pixel camera



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

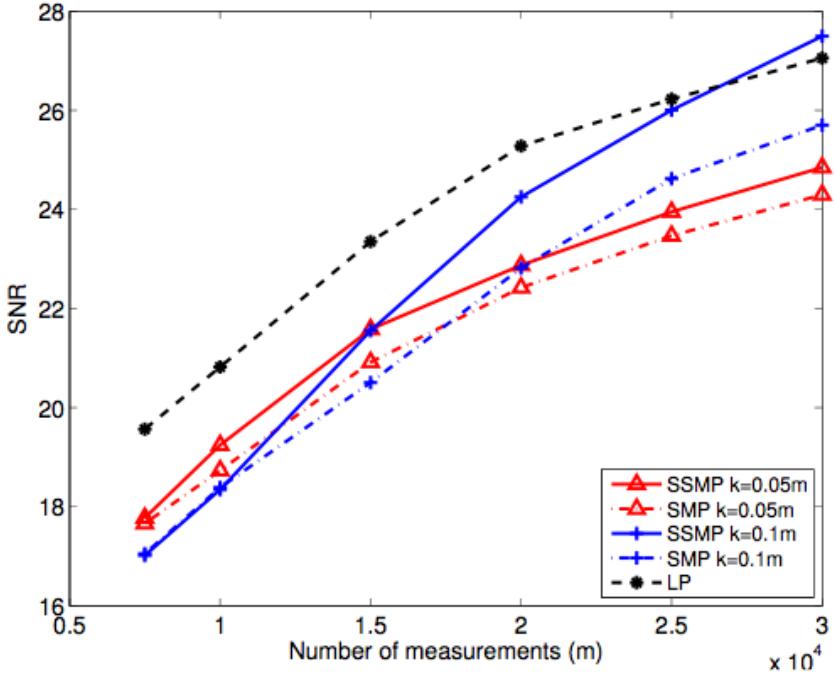
- Pooling Experiments

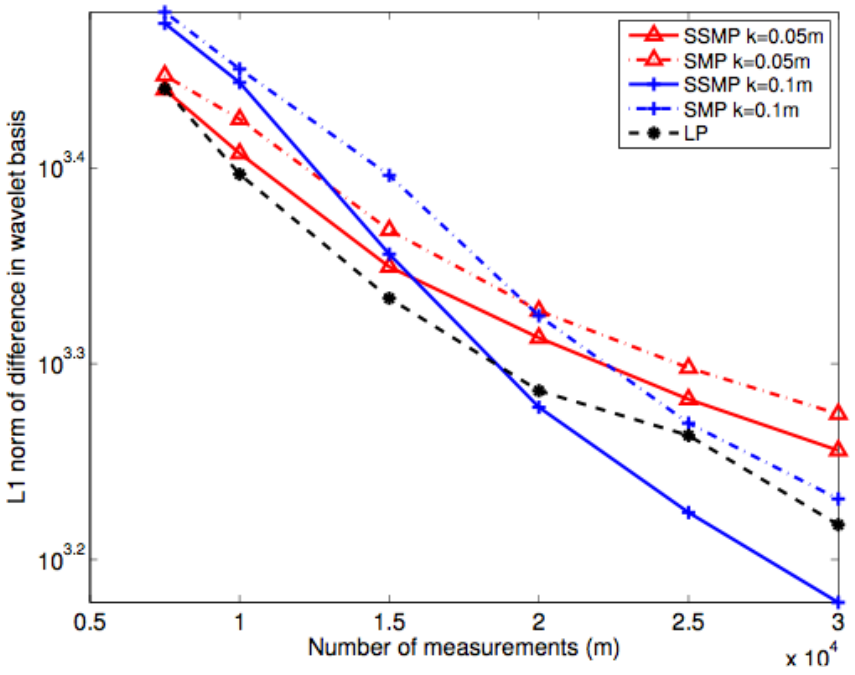
  [Kainkaryam, Woolf'08], [Hassibi et al'07], [Dai-Sheikh, Milenkovic, Baraniuk], [Shental-Amir-Zuk'09],[Erlich-Shental-Amir-Zuk'09]



Experiments

256x256





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

SSMP: Running time

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

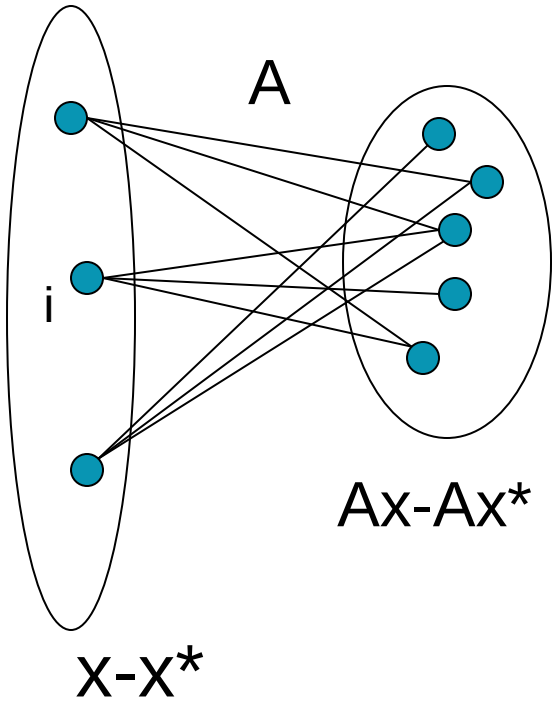
      $$D_i=\min_z \|A(x^*+ze_i)-b\|_1$$

      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:

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

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

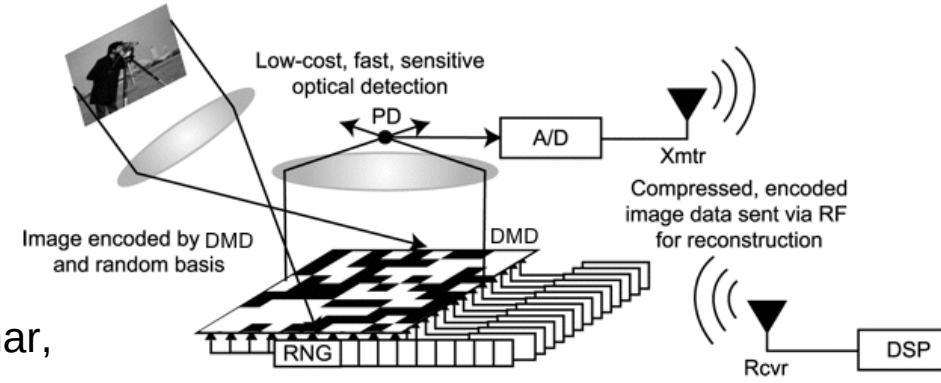


Comments

- Implementing S:
  - Choose a hash function h: {1..m} -> {1..T}
  - Define S={i: h(i)=1}
  - We have $i \in S$ iff h(i)=1

Applications, ctd.

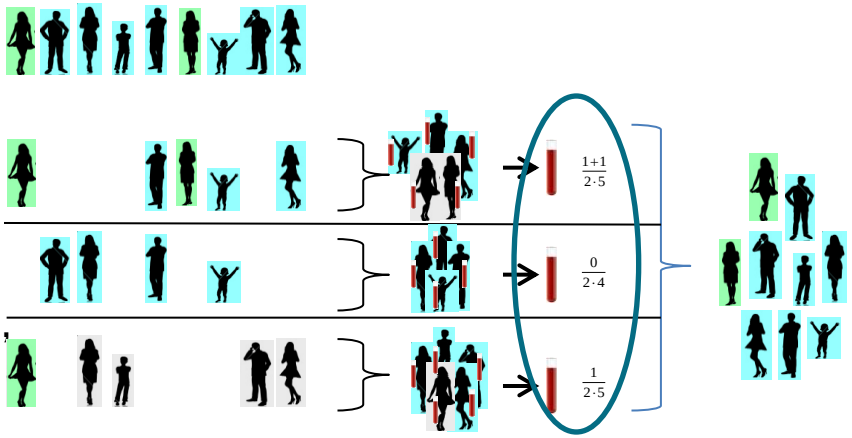
- Single pixel camera



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

- Pooling Experiments

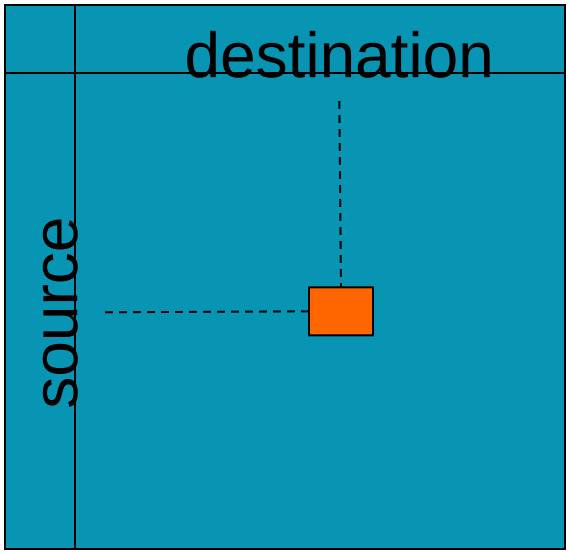
[Kainkaryam, Woolf'08], [Hassibi et al'07], [Dai-Sheikh, Milenkovic, Baraniuk], [Shental-Amir-Zuk'09],[Erlich-Shental-Amir-Zuk'09], [Kainkaryam, Bruex, Gilbert, Woolf'10]



[Shental-et al'09]

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