

Faster Algorithms for Sparse Fourier Transform

Piotr Indyk

MIT

Material from:

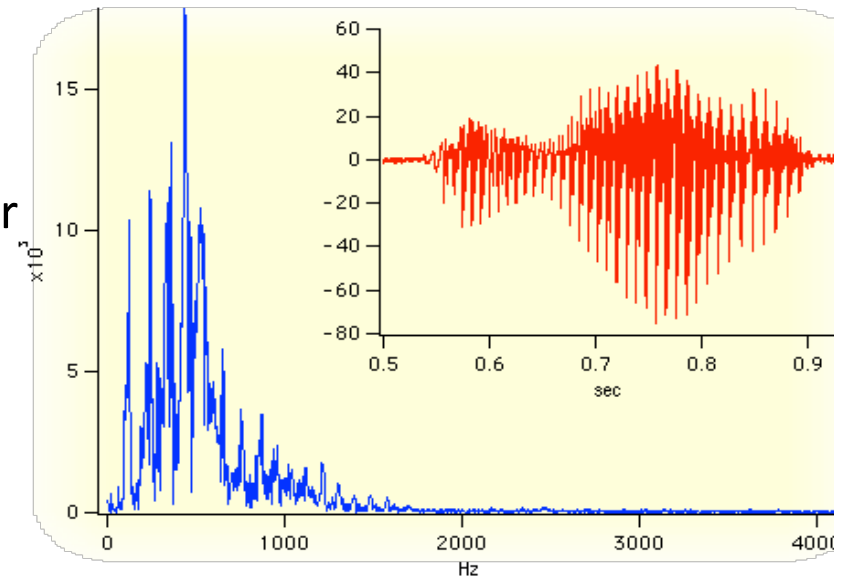
- Hassanieh, Indyk, Katabi, Price, "Simple and Practical Algorithms for Sparse Fourier Transform, SODA'12.
- Hassanieh, Indyk, Katabi, Price, "Nearly Optimal Sparse Fourier Transform", STOC'12.
- Hassanieh, Adib, Katabi, Indyk, "Faster GPS Via the Sparse Fourier Transform", MOBICOM'12
- Ghazi, Hassanieh, Indyk, Katabi, Price, Lixin, "Sample-Optimal Average-Case Sparse Fourier Transform in 2D

Fourier Transform

- Discrete Fourier Transform:
 - Given: a signal $x[1\dots n]$
 - Goal: compute the frequency vector x' where

$$x'_f = \sum_t x_t e^{-2\pi i t f/n}$$

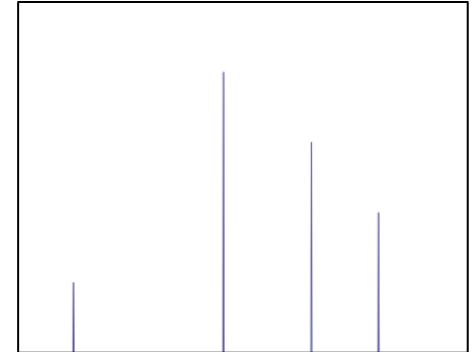
- Very useful tool:
 - Compression (audio, image, video)
 - Signal processing
 - Data analysis
 - Communication
 - Computation (convolution, error-correcting codes, ..)
 - ...



— Sampled Audio Data (Time)
— DFT of Audio Samples (Frequency)

Known algorithms

- Fast Fourier Transform (FFT) computes the frequencies in time $O(n \log n)$
- But, we can do better if we only care about small number k of “dominant frequencies”
 - E.g., recover assume it is k -sparse (only k non-zero entries)
- Algorithms:
 - Boolean cube (Hadamard Transform): [KM] (cf. [GL])
 - Complex FT: [Mansour’92, GGIMS’02, AGS’03, GMS’05, Iwen’10, Akavia’10]
- Best running time: $k \log^c n$ for some $c=O(1)$ [GMS05, Iwen’10]
 - Improve over FFT for $n/k \gg \log^{c-1} n$
- Problem:
 - c is somewhere between 3 and 4
 - Need $n/k > 40,000$ to beat FFTW for $n=2^{22}$
- Goal:
 - Theory: improve over FFT for **all** values of $k=o(n)$
 - Improve in practice



*Assuming entries of x are integers with $O(\log n)$ bits of precision.

Our results: theory

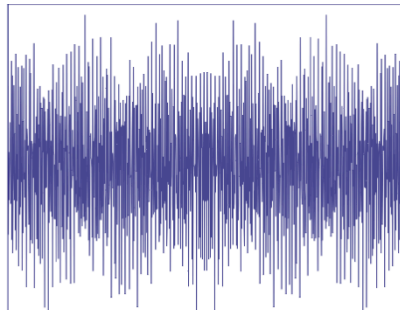
- All algorithms randomized, with constant probability of success, n is a power of 2
- Exactly k -sparse case : $O(k \log n)$
 - Optimal if FFT optimal for $k > n^{\Omega(1)}$
- Approximately k -sparse case - l_2/l_2 guarantee:
$$\|x' - y'\|_2 \leq C \min_{k\text{-sparse } z'} \|x' - z'\|_2$$
 - We get $O(k \log(n) \log(n/k))$ time
 - Improves over FFT for any $k \ll n$
- Slower (but sub-linear) algorithm for a stronger l_∞/l_2 guarantee
- Same time, sample complexity reduced by $\log n$ factor (i.e., to $O(k)$ or $O(k \log(n))$), for the **average case in 2D**
 - Sample optimal (even in the average case)

Our results: experiments

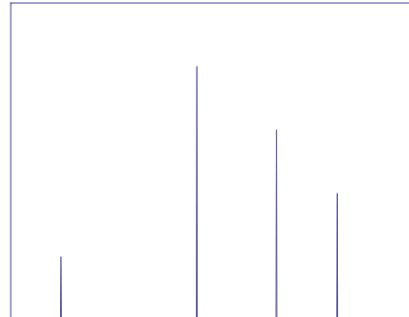
- Significant improvement in running times over prior work
 - E.g., for $n=2^{22}$, a variant of our algorithm (for the exactly k -sparse case) is faster than FFTW for k up to about 2^{17}
 - Best prior implementation of [GMS'05] due to Iwen achieved this breakpoint for k up to about 2^7
- Applications:
 - GPS synchronization [Hassanieh-Adib-Katabi-Indyk'12]
 - Spectrum sensing [Yenduri-Gilbert'12], [Hassanieh-Shi-Abari-Hamed-Katabi'13]
 - Magnetic Resonance Spectroscopy [Shi-Andronesi- Hassanieh-Ghazi, Katabi-Adalsteinsson'13]
 - Exploiting Sparseness in Speech for Fast Acoustic Feature Extraction [Nirjon-Dickerson- Stankovic-Shen-Jiang'13]

Sparse FFT – exact sparsity

Intuition I: Fourier



Time Domain Signal



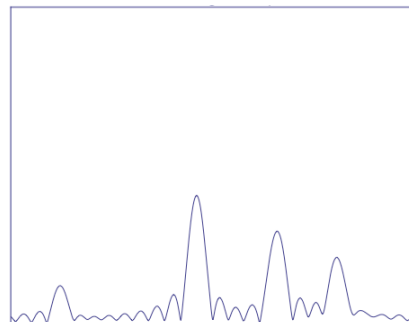
Frequency Domain

n-point DFT : $n \log(n)$

$$\mathbf{x} \longrightarrow \mathbf{x}'$$



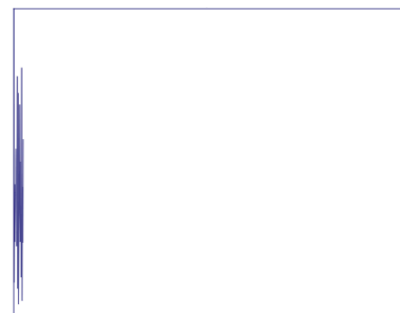
Cut off Time signal



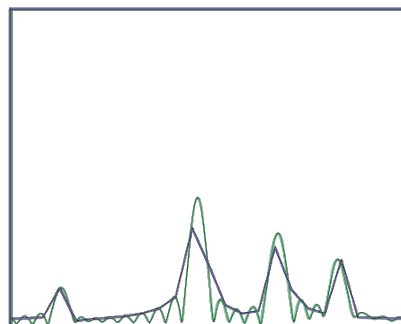
Frequency Domain

n-point DFT of first B terms : $n \log(n)$

$$\mathbf{x} \times \text{Boxcar} \longrightarrow \mathbf{x}' * \text{sinc}$$



First B samples



Frequency Domain

B-point DFT of first B terms: $B \log(B)$

Alias ($\mathbf{x} \times \text{Boxcar}$)



Subsample ($\mathbf{x}' * \text{sinc}$)

Intuition II: Sketching via Hashing*

- Sparse recovery via hashing:
 - x is sparse in the **standard** basis
 - Want to recover x from linear measurements Ax , A is $m \times n$ matrix
 - Prepare a random hash function

$$h: \{1..m\} \rightarrow \{1..B\}$$

- Maintain an array $z = [z_1, \dots, z_B]$ such that

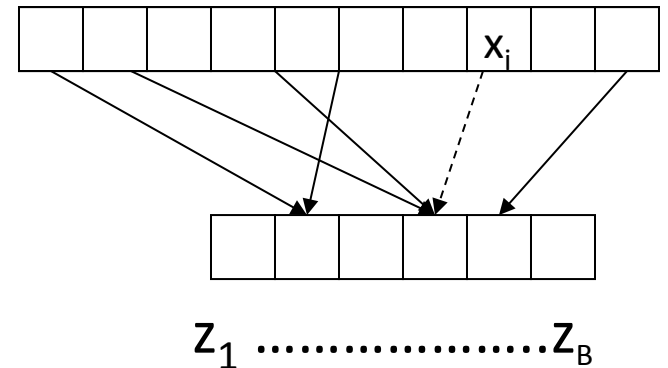
$$z_j = \sum_{i: h(i)=j} x_i$$

- [Eppstein-Goodrich'07]:
 - Suppose x is exactly sparse
 - Also maintain an array $z^* = [z^*_1, \dots, z^*_B]$ such that

$$z^*_j = \sum_{i: h(i)=j} i x_i$$

- If a bucket j has only one heavy coefficient i , we can recover it:

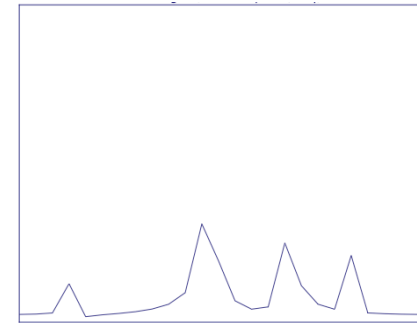
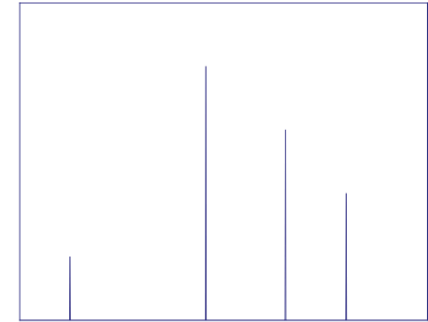
$$i = z^*_j / z_j \text{ and } x_i = z_j$$



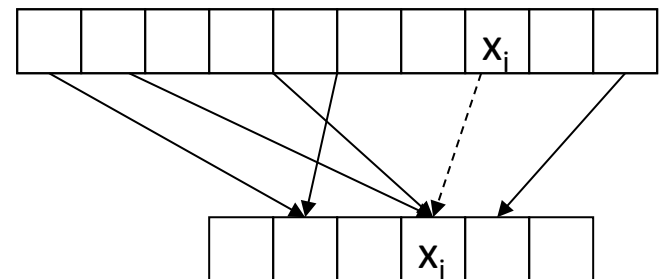
*Charikar-Chen-FarachColton'02, Estan-Varghese'03, Cormode-Muthukrishnan'04, Gilbert-Strauss-Vershynin-Tropp'06, Berinde-Gilbert-Indyk-Karloff-Strauss'08, Sarvotham-Baron-Baraniuk'06,'08, Lu-Montanari-Prabhakar'08, Akcakaya-Tarokh'11....

Main task

- We we would like this

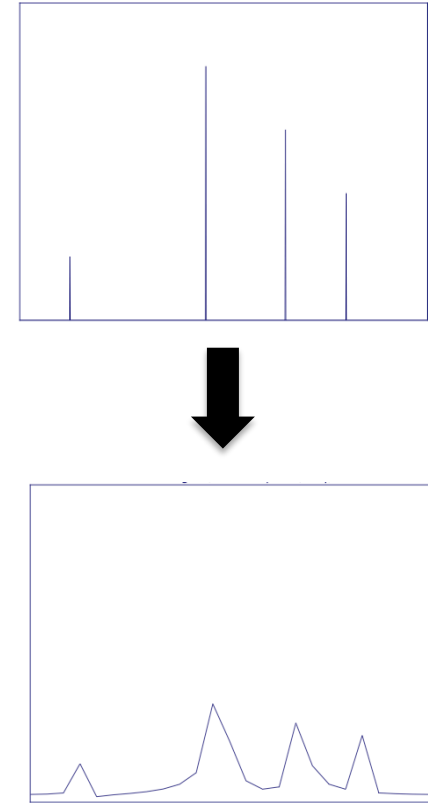


... to act like this:



Issues

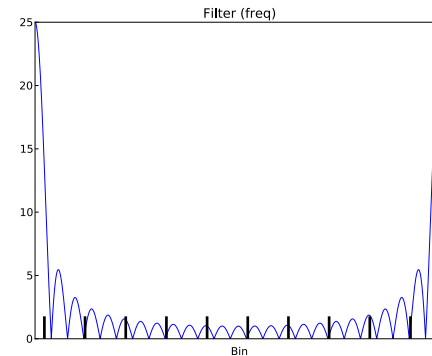
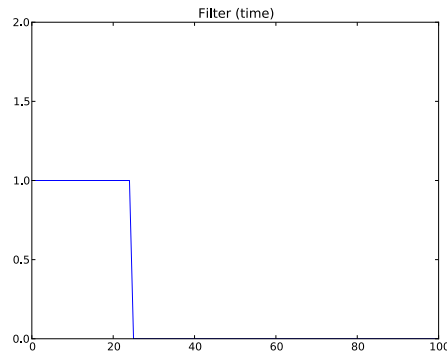
- Can “hash” the Fourier coefficients into B buckets in $O(B \log B)$ time
- Issues:
 - “Hashing”: needs a random hashing of the spectrum
 - Can permute the spectrum pseudo-randomly by permuting the signal [GGIMS’02,GMS’05]
 - “Leaky” buckets
 - Only a small fraction of coefficients collide
 - Subtract and recurse on residual
 - Finding the support
 - Dealing with collisions



Reducing leakage

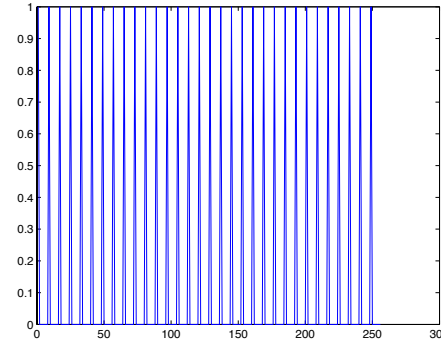
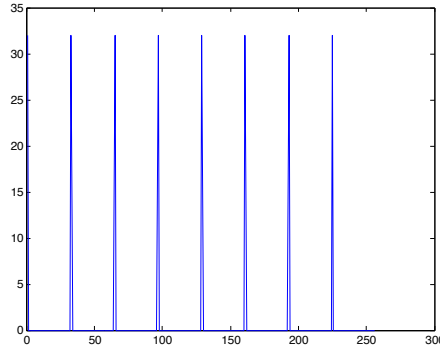


Filters: boxcar filter (used in[GGIMS02,GMS05])



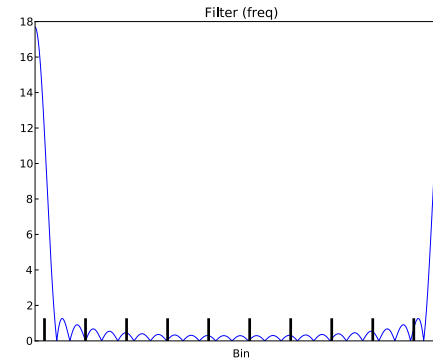
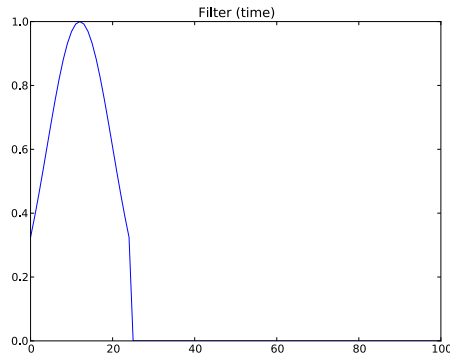
- Boxcar \rightarrow Sinc
 - Polynomial decay
 - Leaking many buckets

Filters: spike train [Man'92,Iwen'10]



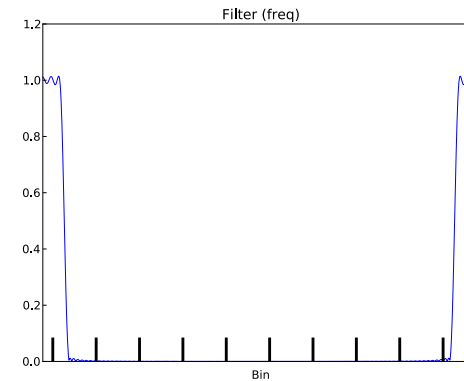
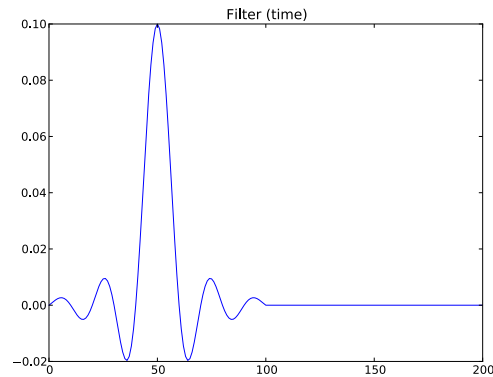
- Zero leakage
- However, hashing does not work
(buckets invariant under hashing, see later)

Filters: Gaussian



- Gaussian -> Gaussian
 - Exponential decay
 - Leaking to $(\log n)^{1/2}$ buckets

Filters: Sinc \times Gaussian

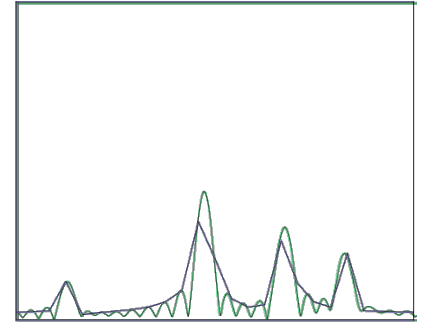


- Sinc \times Gaussian \rightarrow Boxcar*Gaussian
 - Still exponential decay
 - Leaking to <1 buckets
 - Sufficient contribution to the correct bucket
- Actually we use Dolph-Chebyshev filters

Finding the support

Finding the support

- $y' = \text{B-point DFT}(x \times F)$
 $= \text{Subsample}(x' * F')$
- Assume no collisions:
 - At most one large frequency hashes into each bucket.
 - Large frequency f_1 hashes to bucket b_1



$$y'_{b_1} = x'_{f_1} F'_{\Delta} + \text{leakage}$$

- Let x^τ be the signal time-shifted by τ ,
 i.e. $x^\tau_t = x_{t-\tau}$
- Recall $\text{DFT}(x^\tau)_f = x'_f e^{-2\pi i \tau f/n}$
- $y^{\tau'} = \text{B-point DFT}(x^\tau \times F)$

$$y^{\tau'}_{b_1} = x'_{f_1} e^{-2\pi i \tau f_1/n} F'_{\Delta} + \text{leakage}$$

Finding the support, ctd

- At most one non-zero frequency f_1 per bucket b_1
- We have

$$y'_{b_1} = x'_{f_1} F'_{\Delta}$$

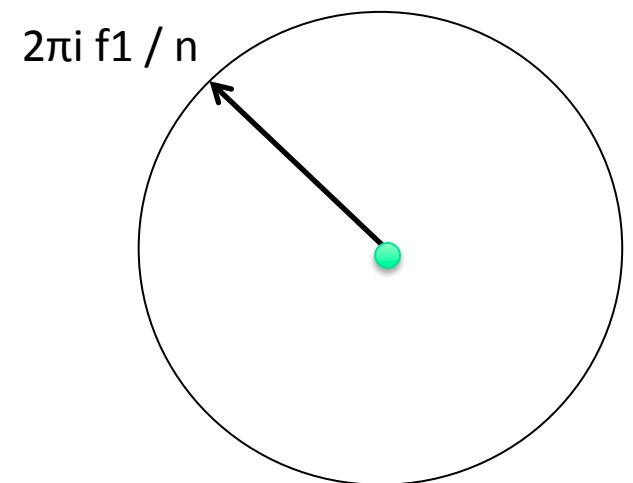
and

$$y'^{\tau}_{b_1} = x'_{f_1} e^{-2\pi i \tau f_1 / n} F'_{\Delta}$$

- So, for $\tau=1$ we have

$$y'_{b_1} / y'^1_{b_1} = e^{-2\pi i f_1 / n}$$

- Can get f_1 from the phase
- **Digression:**
 - Cannot do this when the noise too large (approximately k -sparse case)
 - Instead, read bit by bit, multiply the runtime and sample complexity by $\log(n/k)$



Spectrum Hashing

(used in [GGIMS02, GMS05])

- Every iteration needs new random hashing:
 - Permute time domain signal \rightarrow permute frequency domain
 - Let

$$z_t = x_{\sigma t} e^{-2\pi i t \beta/n}$$

- If σ is invertible mod n

$$z'_f = x'_{1/\sigma f + \beta}$$



Algorithm

(exactly k-sparse case)

- Iteration i :
 1. Set the number of buckets $B_i \approx k/2^{i-1}$
 2. Permute spectrum : $z_t = x_{ot} e^{-2\pi i f \beta/n}$
 3. $y' = B_i$ -point DFT ($z \times F$) = Subsample($z' * F'$)
 4. Repeat with time shift to get y'^τ
 5. Subtract large frequencies recovered in previous iterations
 6. Recover locations and values of remaining large frequencies
- Iteration i recovers $k/2^{i-1}$ of the large frequencies with probability $3/4$ in $O(B_i \log n)$ time
- Total time $O(k \log n)$
 - Steps 3,4 dominated by $B_1=k$
 - Step 5 takes $O(k)$ time per each of the $O(\log n)$ iterations

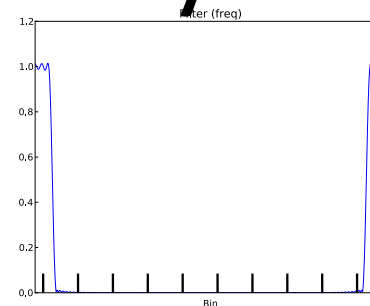
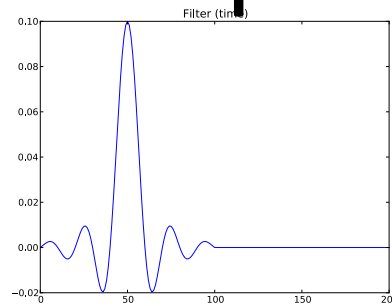
Future directions

Question 1: Sample complexity

Algorithm	Time	Samples	Lower bound
SFFT 3.0 (exact)	$O(k \log n)$	$O(k \log n)$ 	$O(k)$
SFFT 4.0 (robust)	$O(k \log(n) \log(n/k))$	$O(k \log(n) \log(n/k))$ 	$O(k \log(n/k))$

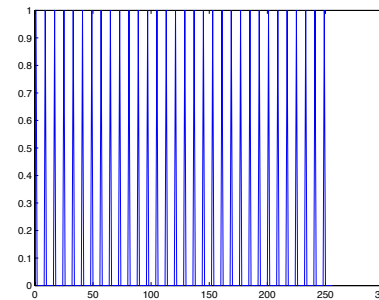
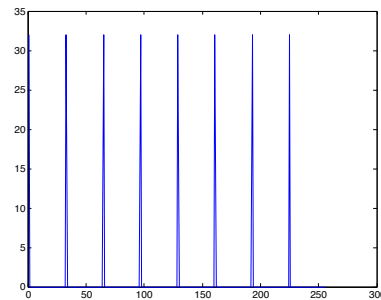
- Can match the lower bound for **average-case** sparsity patterns for 2D Fourier [Ghazi, Hassanieh, Indyk, Katabi, Price, Lixin'13]
- Optimality in the worst-case ?

Sample complexity ctd.



- For $F' = \text{Sinc} \times \text{Gaussian}$

$$|\text{supp}^{1/n^2}(F)| \times |\text{supp}^{1/n^2}(F')| = O(n \log n)$$
- This gives $O(k \log n)$ sample complexity even for the first iteration



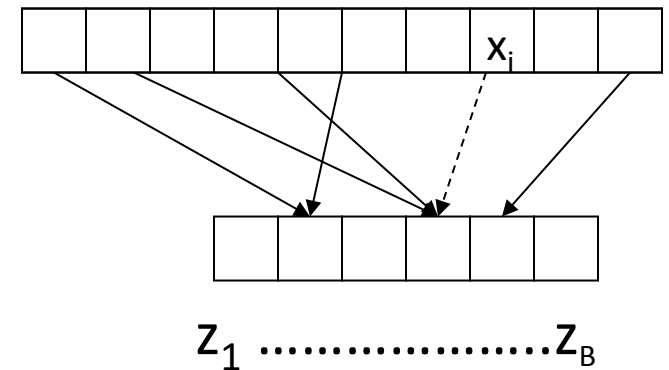
- Spike train is better (optimal): $|\text{supp}(F)| \times |\text{supp}(F')| = n$
- However, can't hash multiple times
 - If several coefficients fall to one bucket under one affine permutation, they do it under all permutations
- **Question:** suppose that n is prime (so no spike trains). Is there a non-trivial* F such that

$$|\text{supp}^{1/n^2}(F)| \times |\text{supp}^{1/n^2}(F')| = o(n \log n) ?$$

*E.g., $|\text{supp}^{1/n}(F)| = \Theta(n^{1/2})$

Question 2: Uniform bounds

- Suppose we would like a sampling pattern that works for all x
- By [Candes-Tao, Rudelson-Vershynin] we know that $O(k \log^4 n)$ samples suffice
 - However, the recovery time is $n \text{polylog } n$ (e.g., CoSaMP)
- Fastest deterministic sub-linear time algorithm has $k^2 \text{polylog } n$ complexity [Iwen]
 - Mimics the bounds achievable for RIP using sparse matrices
- **Question:** can we get $k^{2-a} \text{polylog } n$ bound for some $a > 0$?



Question 3: Higher dimension

- The higher dimension, the sparser the data
- Alas, in d -dimension, the complexity is $O(k (\log n)^{d+1})$
- **Question:** Improve to $O(k \log(n^{d+1}))$?

Conclusions

- $O(k \log n)$ times/samples achievable for the k -sparse case
- $O(k \log n \log(n/k))$ achievable for the L_2/L_2 guarantee
- Questions:
 - Fewer samples (worst case)
 - Higher dimensions
 - Uniform

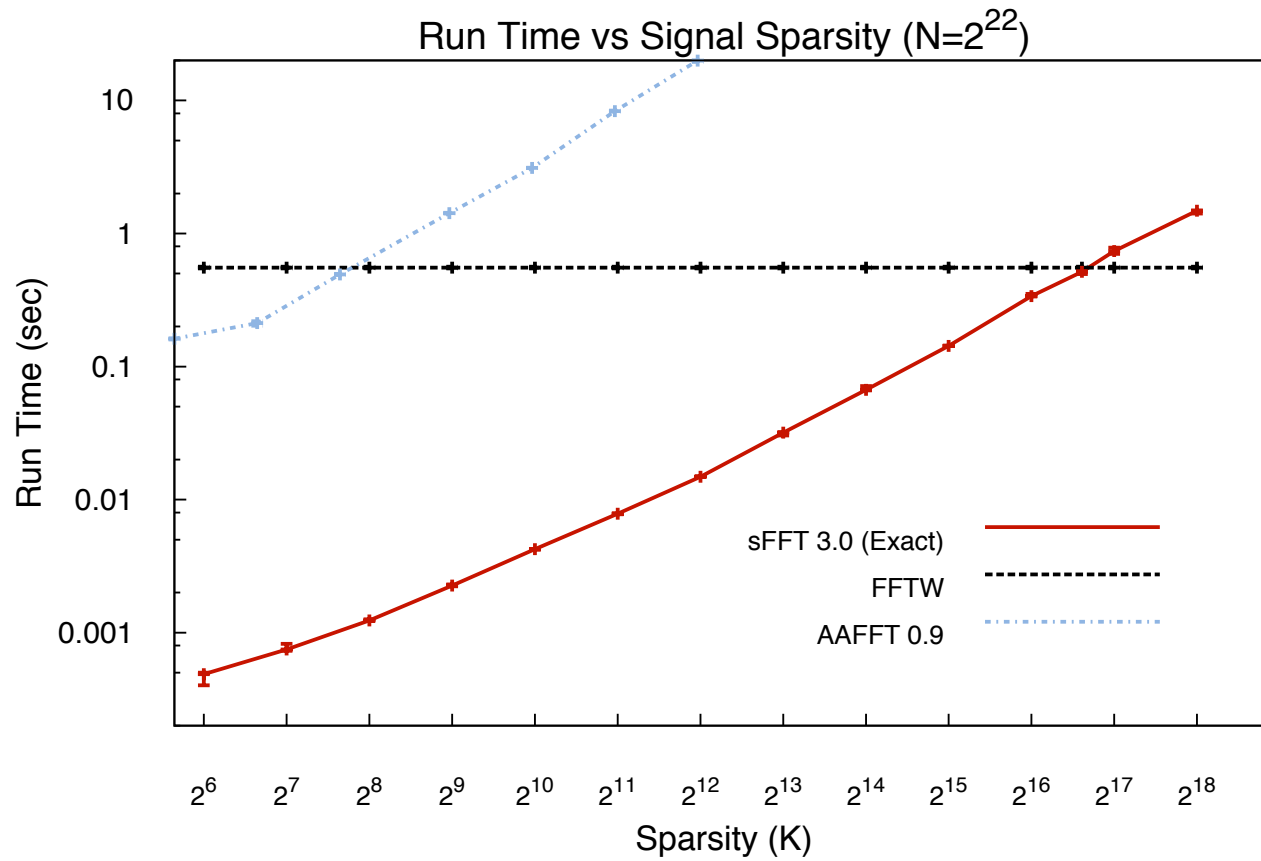
Experiments

(variant of the exactly k -sparse algo)

Setup

- Similar to earlier work:
 - Random 0-1 k -sparse vectors x'
 - Fix n , vary k
 - Fix k , vary n

Experiments



Experiments, ctd

