Sublinear Time, Measurement-Optimal, Sparse Recovery For All

Martin J. Strauss

University of Michigan

Joint with Ely Porat, Bar Ilan
Outline

1 Preliminaries
   - Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   - Avoid lookup table
   - Faster runtime

5 Network Coding—wake up!
   - Conclusion

6 Conclusion
Sparse recovery

\[ \hat{x} = R(\Phi x + \nu) \approx x \]

Approximate best \( k \)-term signal; length is \( N \)
Some criteria of algorithms

Speed
Some criteria of algorithms

- Speed
- Accuracy
Some criteria of algorithms

Number of measurements

Speed

Accuracy
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \left( \binom{N}{k} \right)$
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
  - want $\text{poly}(k \log N)$.
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
  - want $\text{poly}(k \log N)$.
  - Faster than previous *measurement-optimal* algorithms.
Some criteria of algorithms

- **Number of measurements:** want $O(k \log N/k) \approx \log \binom{N}{k}$
- **Recovery runtime (speed):**
  - want $\text{poly}(k \log N)$.
  - Faster than previous *measurement-optimal* algorithms.
  - (“sublinear time” algs lose to “superlinear” FFT every time.)
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
  - want $\text{poly}(k \log N)$.
  - Faster than previous *measurement-optimal* algorithms.
  - (“sublinear time” algs lose to “superlinear” FFT every time.)
- Accuracy—how much error, which norm, universality...our model:
Some criteria of algorithms

- Number of measurements: want $O(k \log N / k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
  - want $\text{poly}(k \log N)$.
  - Faster than previous \textit{measurement-optimal} algorithms.
  - (“sublinear time” algos lose to “superlinear” FFT every time.)
- Accuracy—how much error, which norm, universality...our model:
  - Recover \textit{all} signals in (smaller) $\ell_1$ ball, by \textit{one} matrix.
Some criteria of algorithms

- Number of measurements: want $O(k \log N/k) \approx \log \binom{N}{k}$
- Recovery runtime (speed):
  - want $\text{poly}(k \log N)$.
  - Faster than previous measurement-optimal algorithms.
  - (“sublinear time” algos lose to “superlinear” FFT every time.)
- Accuracy—how much error, which norm, universality...our model:
  - Recover all signals in (smaller) $\ell_1$ ball, by one matrix.
- Norm of error
Some criteria of algorithms

- **Number of measurements**: want $O(k \log N/k) \approx \log \binom{N}{k}$
- **Recovery runtime (speed)**:
  - want $\text{poly}(k \log N)$.
  - Faster than previous *measurement-optimal* algorithms.
  - (“sublinear time” algos lose to “superlinear” FFT every time.)
- **Accuracy**—how much error, which norm, universality...our model:
  - Recover *all* signals in (smaller) $\ell_1$ ball, by one matrix.
- **Norm of error**
  - Want $\ell_2$:
    $$\|x - \hat{x}\|_2 \leq \frac{\epsilon}{\sqrt{k}} \|x - x_k\|_1.$$
Some criteria of algorithms

- **Number of measurements**: want $O(k \log N/k) \approx \log \binom{N}{k}$
- **Recovery runtime (speed)**:
  - want $\text{poly}(k \log N)$.
  - Faster than previous *measurement-optimal* algorithms.
  - ("sublinear time" algos lose to "superlinear" FFT every time.)
- **Accuracy**—how much error, which norm, universality...our model:
  - Recover *all* signals in (smaller) $\ell_1$ ball, by *one* matrix.
- **Norm of error**
  - Want $\ell_2$:
    $$\|x - \hat{x}\|_2 \leq \frac{\epsilon}{\sqrt{k}} \|x - x_k\|_1.$$
  - Here get only $\ell_1$ (strictly worse):
    $$\|x - \hat{x}\|_1 \leq (1 + \epsilon) \|x - x_k\|_1.$$
### Some results

<table>
<thead>
<tr>
<th>Paper</th>
<th>No. meas.</th>
<th>time</th>
<th>norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GSTV07]</td>
<td>$k\ \text{polylog}$</td>
<td>$\text{poly}(k \log N)$</td>
<td>2</td>
</tr>
<tr>
<td>[Donoho04]</td>
<td>$k \log(N/k)$</td>
<td>$\text{poly}(N)$</td>
<td>2</td>
</tr>
<tr>
<td>[CRT04]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[RI08]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Red is optimal.
## Some results

<table>
<thead>
<tr>
<th>Paper</th>
<th>No. meas.</th>
<th>time</th>
<th>norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GSTV07]</td>
<td>$k \text{ polylog}$</td>
<td>$\text{poly}(k \log N)$</td>
<td>2</td>
</tr>
<tr>
<td>[Donoho04]</td>
<td>$k \log(N/k)$</td>
<td>$\text{poly}(N)$</td>
<td>2</td>
</tr>
<tr>
<td>[CRT04]</td>
<td>$k \log(N/k)$</td>
<td>$N \log(N/k)$</td>
<td></td>
</tr>
<tr>
<td>[RI08]</td>
<td>$k \log(N/k)$</td>
<td>$N \log(N/k)$</td>
<td>1</td>
</tr>
</tbody>
</table>

Red is optimal.
## Some results

<table>
<thead>
<tr>
<th>Paper</th>
<th>No. meas.</th>
<th>time</th>
<th>norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GSTV07]</td>
<td>(k) polylog</td>
<td>(\text{poly}(k \log N))</td>
<td>2</td>
</tr>
<tr>
<td>[Donoho04]</td>
<td>(k \log(N/k))</td>
<td>(\text{poly}(N))</td>
<td>2</td>
</tr>
<tr>
<td>[CRT04]</td>
<td>(k \log(N/k))</td>
<td>(N \log(N/k))</td>
<td>1</td>
</tr>
<tr>
<td>[RI08]</td>
<td>(k \log(N/k))</td>
<td>(\text{poly}(k \log N))</td>
<td>1</td>
</tr>
<tr>
<td>Here/In progress</td>
<td>(k \log(N/k))</td>
<td>(\text{poly}(k \log N))</td>
<td>1</td>
</tr>
</tbody>
</table>

Red is optimal.

Here is [http://arxiv.org/abs/1012.1886v2](http://arxiv.org/abs/1012.1886v2)
Some results

<table>
<thead>
<tr>
<th>Paper</th>
<th>No. meas.</th>
<th>time</th>
<th>norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GSTV07]</td>
<td>$k \ \text{polylog}$</td>
<td>$\text{poly}(k \log N)$</td>
<td>2</td>
</tr>
<tr>
<td>[Donoho04]</td>
<td>$k \log(N/k)$</td>
<td>$\text{poly}(N)$</td>
<td>2</td>
</tr>
<tr>
<td>[CRT04]</td>
<td>$k \log(N/k)$</td>
<td>$N \log(N/k)$</td>
<td>1</td>
</tr>
<tr>
<td>[RI08]</td>
<td>$k \log(N/k)$</td>
<td>$\text{poly}(k \log N)$</td>
<td>1</td>
</tr>
<tr>
<td>Here/In progress</td>
<td>$k \log(N/k)$</td>
<td>$\text{poly}(k \log N)$</td>
<td>1</td>
</tr>
</tbody>
</table>

Red is optimal.

Here is http://arxiv.org/abs/1012.1886v2

Other models by: Xu-Hassibi, Caldebank-Howard-Jafarpour, Gilbert-Li-P-S, ...
Group testing 1-sparse signals

Group testing on 1-sparse signal. First half or second? Recover bit-by-bit:

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\]
Group testing 1-sparse signals

Group testing on 1-sparse signal. First half or second? Recover bit-by-bit:
Group testing 1-sparse signals

Group testing on 1-sparse signal. First half or second? Recover bit-by-bit:

\[
\begin{pmatrix}
7 \\
0 \\
7 \\
0
\end{pmatrix} \approx \begin{pmatrix}
\text{Reference} \\
\text{Small} \\
\text{BIG} \\
\text{Small}
\end{pmatrix} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix} \cdot \begin{pmatrix}
\text{noise} \\
\text{noise} \\
\text{noise} \\
\text{noise} \\
\text{noise} \\
\text{noise} \\
\text{noise} \\
\text{noise}
\end{pmatrix}
\]
Techniques for some sublinear algorithms

- Hash into $k$ buckets (hope to isolate HH’s with low noise)

\[ H = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \]

- Group testing on 1-sparse signal.

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{pmatrix}
\]

- Typically lose log factor in meas. Top row of $H$ becomes:

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{pmatrix}
\]
1 Preliminaries
   • Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   • Avoid lookup table
   • Faster runtime

5 Network Coding—wake up!
   • Conclusion

6 Conclusion
Algorithm

- Hash into $B = \sqrt{kN}$ buckets;

![Algorithm Diagram](image-url)
Hash into $B = \sqrt{kN}$ buckets; Aggregate;

$\text{Time} \approx k \log \frac{B}{k}$

$\text{Time} \approx k \log \frac{N}{k}$
Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure

```
Hash
Aggregate
Measure
```

$\approx k, B$

$k \log(B/k)$

$\approx k, N$

$Lift solution (from table).

$Time \approx \text{no. preimages} = k \left( \frac{N}{B} \right) = \sqrt{kN}$

$\text{collect measurements}$

$\text{Recursively solve, naively.}$

$Time \approx \text{length} = B = \sqrt{kN}$

$\text{Repeat } \log\left( \frac{N}{k} \right) / \log\left( \frac{B}{k} \right) = 2$ times.
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k)/\log(B/k) = 2$ times;
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k) / \log(B/k) = 2$ times; collect measurements
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k)/\log(B/k) = 2$ times; collect measurements
- Recursively solve, naively.
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k)/\log(B/k) = 2$ times; collect measurements
- Recursively solve, naively. Time $\approx$ length $= B = \sqrt{kN}$
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k)/\log(B/k) = 2$ times; collect measurements
- Recursively solve, naively. Time $\approx$ length $= B = \sqrt{kN}$
- Lift solution (from table).

```
Hash
Aggregate
Measure
Collect
```

```
(k, N)
(\approx k, B)
k \log(B/k)
```

```
k \log(N/k)
```

Martin J. Strauss (University of Michigan)
Algorithm

- Hash into $B = \sqrt{kN}$ buckets; Aggregate; Measure
- Repeat $\log(N/k)/\log(B/k) = 2$ times; collect measurements
- Recursively solve, naively. Time $\approx$ length $= B = \sqrt{kN}$
- Lift solution (from table). Time $\approx$ no. preimages $= k(N/B) = \sqrt{kN}$
Outline

1 Preliminaries
   ● Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   ● Avoid lookup table
   ● Faster runtime

5 Network Coding—wake up!
   ● Conclusion

6 Conclusion
Result

Theorem

Algorithm takes $\approx \sqrt{kN}$ time and uses $k \log(N/k)$ measurements.
Result and Analysis

Result

Theorem

*Algorithm takes* \( \approx \sqrt{kN} \) *time and uses* \( k \log(N/k) \) *measurements.*

Need to show:

- Number of measurements and runtime—done.
- Correctness of Hashing procedure
  - Why \( 2 = \log(N/k)/\log(B/k) \) repetitions?
  - Why do we get \( (\approx k, B) \)-signal?
- Correctness of recursive solution—easy
- Correctness of lifting—easy by (lazy) design (use of table)
Correctness of Hashing

Lemma

*Intermediate signal is indeed ≈ k-sparse and length B.*
Correctness of Hashing

Lemma

*Intermediate signal is indeed $\approx k$-sparse and length $B$.*

- Each heavy hitter is isolated except with prob $k/B$. 

Failure probs drop to $(k/N)^k \leq (N^k - 1)$ and $(k/N)^t \leq (N^t - 1)$ when repeating log($N/k$) / log($B/k$) times.
Correctness of Hashing

Lemma

*Intermediate signal is indeed \( \approx \) \( k \)-sparse and length \( B \).*

- Each heavy hitter is isolated except with prob \( k/B \).
  - \( \geq k/2 \) fail with prob \( (k/B)^k = 2^{-k \log(B/k)} \)
Correctness of Hashing

Lemma

*Intermediate signal is indeed $\approx k$-sparse and length $B$.*

- Each heavy hitter is isolated except with prob $k/B$.
  - $\geq k/2$ fail with prob $(k/B)^k = 2^{-k \log(B/k)}$
- Heavy hitters land in set $S$ of about $k$ of $B$ buckets. Consider $t$ noise items of size $1/t$:
Correctness of Hashing

Lemma

*Intermediate signal is indeed $\approx k$-sparse and length $B$.*

- Each heavy hitter is isolated except with prob $k/B$.
  - $\geq k/2$ fail with prob $(k/B)^k = 2^{-k\log(B/k)}$
- Heavy hitters land in set $S$ of about $k$ of $B$ buckets. Consider $t$ noise items of size $1/t$:
  - Each noise item lands in $S$ with prob $k/B$
Correctness of Hashing

Lemma

*Intermediate signal is indeed $\approx k$-sparse and length $B$.*

- Each heavy hitter is isolated except with prob $k/B$.
  - $\geq k/2$ fail with prob $(k/B)^k = 2^{-k \log(B/k)}$

- Heavy hitters land in set $S$ of about $k$ of $B$ buckets. Consider $t$ noise items of size $1/t$:
  - Each noise item lands in $S$ with prob $k/B$
  - $\geq t/2$ noise items land in $S$ with prob $(k/B)^t = 2^{-t \log(B/k)}$
    (Otherwise, enough of $S$ survives)
Correctness of Hashing

Lemma

Intermediate signal is indeed $\approx k$-sparse and length $B$.

- Each heavy hitter is isolated except with prob $k/B$.
  - $\geq k/2$ fail with prob $(k/B)^k = 2^{-k \log(B/k)}$

- Heavy hitters land in set $S$ of about $k$ of $B$ buckets. Consider $t$ noise items of size $1/t$:
  - Each noise item lands in $S$ with prob $k/B$
  - $\geq t/2$ noise items land in $S$ with prob $(k/B)^t = 2^{-t \log(B/k)}$
    (Otherwise, enough of $S$ survives)

- Repeat $\log(N/k)/\log(B/k)$ times
Correctness of Hashing

Lemma

*Intermediate signal is indeed $\approx k$-sparse and length $B$.*

- Each heavy hitter is isolated except with prob $k/B$.
  - $\geq k/2$ fail with prob $(k/B)^k = 2^{-k \log(B/k)}$

- Heavy hitters land in set $S$ of about $k$ of $B$ buckets. Consider $t$ noise items of size $1/t$:
  - Each noise item lands in $S$ with prob $k/B$
  - $\geq t/2$ noise items land in $S$ with prob $(k/B)^t = 2^{-t \log(B/k)}$
    (Otherwise, enough of $S$ survives)

- Repeat $\log(N/k)/\log(B/k)$ times
  - Failure probs drop to $(k/N)^k \leq \binom{N}{k}^{-1}$ and $(k/N)^t \leq \binom{N}{t}^{-1}$
  - Take union bound.
More generally...

- Cascade through any chosen number $\ell$ of levels.
- $\text{poly}(\ell)$ problems with parameters $(k, k(N/k)^{1/\ell})$
- Time around $\text{poly}(\ell) k(N/k)^{1/\ell}$
- Number of measurements is around $\text{poly}(\ell) k \log(N/k)$
Outline

1 Preliminaries
   • Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   • Avoid lookup table
   • Faster runtime

5 Network Coding—wake up!
   • Conclusion

6 Conclusion
What’s next?

(Joint with Anna Gilbert, Yi Li, Ely Porat)
What’s next?

(Joint with Anna Gilbert, Yi Li, Ely Porat)

- Avoid lookup table
What’s next?

(Joint with Anna Gilbert, Yi Li, Ely Porat)

- Avoid lookup table
- Lower runtime from \(\text{poly}(\ell) k (N/k)^{1/\ell}\) to \(\text{poly}(k, \log N)\)
When we recover heavy hitter $i$, ...can also get arbitrary $O(\log(B/k))$-bit message! (Partially) codes pointer back to $i \in \mathbb{N}$. No need to store back pointer: $B \rightarrow \mathbb{N}$ explicitly in table. Only need to use hash function forwards: $\mathbb{N} \rightarrow B$. 
Avoid lookup table

- When we recover heavy hitter $i$,
  - ...can also get arbitrary $O(\log(B/k))$-bit message!
Avoid lookup table

- When we recover heavy hitter $i$,
  - ...can also get arbitrary $O(\log(B/k))$-bit message!
- (Partially) codes pointer back to $i' \in [N]$. 
Avoid lookup table

- When we recover heavy hitter $i$,
  - ...can also get arbitrary $O(\log(B/k))$-bit message!
- (Partially) codes pointer back to $i' \in [N]$.
  - No need to store back pointer: $[B] \rightarrow [N]$ explicitly in table.
Avoid lookup table

- When we recover heavy hitter $i$,
  - ...can also get arbitrary $O(\log(B/k))$-bit message!

- (Partially) codes pointer back to $i' \in [N]$.
  - No need to store back pointer: $[B] \rightarrow [N]$ explicitly in table.
  - Only need to use hash function \textit{forwards}: $[N] \rightarrow [B]$. 
Sparse recovery channel—The medium is the message

- Encoder and Decoder agree on $\Phi$ (independent of message)
Sparse recovery channel—The medium is the message

- Encoder and Decoder agree on $\Phi$ (independent of message)
- Message $m$ of length $B$ and alphabet size $B/k$
Sparse recovery channel—The medium is the message

- Encoder and Decoder agree on $\Phi$ (independent of message)
- Message $m$ of length $B$ and alphabet size $B/k$
- Encoder makes final measurement matrix $\Phi'$ from $\Phi$ and $m$
Sparse recovery channel—The medium is the message

- Encoder and Decoder agree on \( \Phi \) (independent of message)
- Message \( m \) of length \( B \) and alphabet size \( B/k \)
- Encoder makes final measurement matrix \( \Phi' \) from \( \Phi \) and \( m \)
- Channel picks \( x \) and \( \nu (\neq 0?) \) and produces \( \Phi'x + \nu \).
Sparse recovery channel—The medium is the message

- Encoder and Decoder agree on $\Phi$ (independent of message)
- Message $m$ of length $B$ and alphabet size $B/k$
- Encoder makes final measurement matrix $\Phi'$ from $\Phi$ and $m$
- Channel picks $x$ and $\nu(\neq 0?)$ and produces $\Phi'x + \nu$.
- Decoder tries to recover $m$ in $x$-weighted sense; need $\hat{m}_i \approx m_i$ for many $i$ such that $|x_i|$ is large. (Decoder doesn’t know $x$.)
Coding one bit

(Reduced group testing.) Hash into $k$ buckets. One bucket:

$\begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0
\end{pmatrix}$

1-bit message

$\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}$

Leads to

$\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}$
Coding one bit

(Reduced group testing.) Hash into $k$ buckets. One bucket:

$$
\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
$$

1-bit message

$$
\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
$$

Leads to

$$
\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
$$

- With $k \log(B/k)$ measurements, $\log(B/k)$ lossy chances to code bits.
Coding one bit

(Reduced group testing.) Hash into \( k \) buckets. One bucket:

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

1-bit message

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]

Leads to

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]

- With \( k \log(B/k) \) measurements, \( \log(B/k) \) lossy chances to code bits.
- With ECC, get \( \log(B/k)(\approx \log N?) \)-bit msg for each HH.
Coding one bit

(Reduced group testing.) Hash into $k$ buckets. One bucket:

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

1-bit message

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]

Leads to

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]

- With $k \log(B/k)$ measurements, $\log(B/k)$ lossy chances to code bits.
- With ECC, get $\log(B/k)(\approx \log N)$-bit msg for each HH.

Use message to lift solution, rather than explicit lookup table.
Three kinds of information

Algorithm:
- Hash into $B$ buckets
- Repeat $r = \log(N/k)/\log(N/B)$ times
- Solve recursively

Need $\log N$ bits of backpointer hash$^{-1} : \rightarrow [N]$. 
Three kinds of information

Algorithm:
- Hash into $B$ buckets
- Repeat $r = \log(N/k)/\log(N/B)$ times
- Solve recursively

Need $\log N$ bits of backpointer hash $^{-1} : [N]$. $j$’th repetition, $j = 1, 2, \ldots, r$, gives tuple of
  - $\log(B/k)$ codeable bits $m_i$
  - $j$ (side information)
  - Index $i_j \in [B]$ of recursive heavy hitter in $j$’th repetition ($\log B$ non-codeable bits)
Three kinds of information

Algorithm:
- Hash into $B$ buckets
- Repeat $r = \log(N/k)/\log(N/B)$ times
- Solve recursively

Need $\log N$ bits of backpointer hash$^{-1} : \rightarrow [N]$. 
$j$’th repetition, $j = 1, 2, \ldots, r$, gives tuple of
- $\log(B/k)$ codeable bits $m_i$
- $j$ (side information)
- Index $i_j \in [B]$ of recursive heavy hitter in $j$’th repetition ($\log B$ non-codeable bits)

Code payload and linking information into $m_i$ and assemble.
Three kinds of information

Algorithm:
- Hash into $B$ buckets
- Repeat $r = \log(N/k)/\log(N/B)$ times
- Solve recursively

Need $\log N$ bits of backpointer hash $^{-1} : \rightarrow [N]$.

For the $j$'th repetition, $j = 1, 2, \ldots, r$, gives tuple of
- $\log(B/k)$ codeable bits $m_i$
- $j$ (side information)
- Index $i_j \in [B]$ of recursive heavy hitter in $j$'th repetition ($\log B$ non-codeable bits)

Code payload and linking information into $m_i$ and assemble.

How?
Outline

1 Preliminaries
   - Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   - Avoid lookup table
   - Faster runtime

5 Network Coding—wake up!
   - Conclusion

6 Conclusion
First, network (rateless) coding:

Message (movie) of length $n$ downloaded for later viewing (not streamed, not DVD by mail, ...)

Flaky network—dropped connections (erasures) but no errors

Publisher breaks message into $p$ packets, encodes, and broadcasts continually

Subscriber needs any $O(p)$ packets to recover message.

Punchline, e.g., Send ever-new points on graph of degree-$p$ polynomial. Any $p + 1$ points suffice.
Network coding

First, network (rateless) coding:
- Message (movie) of length $n$
Network coding

First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)

Publisher breaks message into $p$ packets, encodes, and broadcasts continually.

Subscriber needs any $O(p)$ packets to recover message.

Punchline, e.g., Send ever-new points on graph of degree-$p$ polynomial. Any $p + 1$ points suffice.
Network coding

First, network (rateless) coding:
- Message (movie) of length \( n \)
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but *no errors*
Network coding

First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but no errors
- Publisher breaks message into $p$ packets, encodes, and broadcasts continually
Network coding

First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but *no errors*
- Publisher breaks message into $p$ packets, encodes, and broadcasts continually
- Subscriber needs *any* $O(p)$ packets to recover message.
First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but *no errors*
- Publisher breaks message into $p$ packets, encodes, and broadcasts continually
- Subscriber needs *any* $O(p)$ packets to recover message.
- Punchline, e.g.,

Send ever-new points on graph of degree-$p$ polynomial. Any $p+1$ points suffice.
Network coding

First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but no errors
- Publisher breaks message into $p$ packets, encodes, and broadcasts continually
- Subscriber needs any $O(p)$ packets to recover message.
- Punchline, e.g.,
  - Send ever-new points on graph of degree-$p$ polynomial.
Network coding

First, network (rateless) coding:

- Message (movie) of length $n$
- Download for later viewing (not streamed, not DVD by mail, ...)
- Flaky network—dropped connections (erasures) but no errors
- Publisher breaks message into $p$ packets, encodes, and broadcasts continually
- Subscriber needs any $O(p)$ packets to recover message.
- Punchline, e.g.,
  - Send ever-new points on graph of degree-$p$ polynomial.
  - Any $p + 1$ points suffice.
Multiple-stream network coding problem

- Unordered set of $k$ messages (movies), length $n$, transmitted simultaneously.

Total $kn$ bits. Want to recover from $O(nk)$ total bits, avoiding $\log k$ header bits (which movie?) per packet.

Get error correction for free! Why?
Packets from one movie can be regarded as noise in another.
Multiple-stream network coding problem

- Unordered set of $k$ messages (movies), length $n$, transmitted simultaneously.
- Total $kn$ bits. Want to recover from $O(nk)$ total bits, avoiding $\log k$ header bits (which movie?) per packet.
Multiple-stream network coding problem

- Unordered set of $k$ messages (movies), length $n$, transmitted simultaneously.
  Total $kn$ bits. Want to recover from $O(nk)$ total bits, avoiding $\log k$ header bits (which movie?) per packet.

Get error correction for free! Why?
Multiple-stream network coding problem

- Unordered set of \( k \) messages (movies), length \( n \), transmitted simultaneously.
  
  Total \( kn \) bits. Want to recover from \( O(nk) \) total bits, avoiding \( \log k \) header bits (which movie?) per packet.

Get error correction for free! Why?

Packets from one movie
Multiple-stream network coding problem

- Unordered set of $k$ messages (movies), length $n$, transmitted simultaneously.
  Total $kn$ bits. Want to recover from $O(nk)$ total bits, avoiding $\log k$ header bits (which movie?) per packet.

Get error correction for free! Why?

Packets from one movie can be regarded as noise in another.
Upcoming results

Theorem

There’s an algorithm that runs in time $k \log^{O(1)} N$, uses $O(k \log N/k)$ measurements, and returns $\hat{x}$ with

$$\|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1.$$  

(Joint with Anna Gilbert, Yi Li, Ely Porat)
Upcoming results

Theorem

There's an algorithm that runs in time $k \log^{O(1)} N$, uses $O(k \log N/k)$ measurements, and returns $\hat{x}$ with

$$\|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1.$$ 

(Joint with Anna Gilbert, Yi Li, Ely Porat)

Can this be improved with better error-correcting codes?
Outline

1 Preliminaries
   - Problem we are addressing

2 Algorithm

3 Result and Analysis

4 Next Results
   - Avoid lookup table
   - Faster runtime

5 Network Coding—wake up!
   - Conclusion

6 Conclusion
Conclusion

- First sublinear-time algo with optimal measurements in forall model, with

\[ \|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1 \]
Conclusion

- First sublinear-time algo with optimal measurements in forall model, with
  \[ \|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1 \]

- Time \( \sqrt{kN} \), improveable (?) to \( \text{poly}(k, \log N) \)
Conclusion

- First sublinear-time algo with optimal measurements in forall model, with
  \[ \|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1 \]

- Time $\sqrt{kn}$, improveable (?) to $\text{poly}(k, \log N)$
- Lookup table of size $Nk^{1/4}$, removeable (?)
Conclusion

- First sublinear-time algo with optimal measurements in forall model, with
  \[ \|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_k\|_1 \]

- Time \(\sqrt{kn}\), improveable (?) to poly\((k, \log N)\)
- Lookup table of size \(Nk^{1/4}\), removeable (?)

Finale is open: Improve to 2-norm:

\[ \|x - \hat{x}\|_2 \leq \frac{\epsilon}{\sqrt{k}}\|x - x_k\|_1. \]