Network Coding and Index Coding via Rank Minimization

Salim El Rouayheb
IIT, Chicago

Joint work with Xiao Huang
Index Coding

<table>
<thead>
<tr>
<th>Transmission #</th>
<th>Index code 1</th>
<th>Index code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X_1$</td>
<td>$X_1 + X_2$</td>
</tr>
<tr>
<td>2</td>
<td>$X_2$</td>
<td>$X_3$</td>
</tr>
<tr>
<td>3</td>
<td>$X_3$</td>
<td>$X_4$</td>
</tr>
<tr>
<td>4</td>
<td>$X_4$</td>
<td></td>
</tr>
</tbody>
</table>

Informed-source coding-on-demand [Birk & Kol infocom’98]
Index Coding & Graph Coloring

Side info graph $G_d$

User 1 has packet 2

$X_1 + X_2$

Clique cover of $G$

$= \text{Chromatic nbr of } \overline{G}$

$X_3 + X_4$

$X_1 + X_2 + X_3 + X_4$

$\overline{G}$
Index Coding & Graph Coloring

\[
\alpha(G_d) \leq c(G_d) \leq L_{min} \leq \chi_f(\overline{G}) \leq \chi(\overline{G})
\]

Independence nbr

Shannon capacity
[Haemers ‘79]

Fractional Chromatic nbr
[Blasiak et al.‘11]

\[
\leq \chi_f\ell(G)
\]

Fractional local chrom. nbr
[Shanmugan et al. ‘13]

[Alon et al., ‘08]

[Maleki, Cadambe, Jafar ‘12]

[Arbabjolfaei et al., ‘13]

…
Index Coding on Erdős-Rényi Graphs

\[ \alpha(G) \leq L^*_{min} \leq \chi(\overline{G}) \]

- When \( n \to \infty \), we have with prob 1

\[ \log n \leq L^*_{min} \leq \frac{n}{\log n} \]

- Can improve the lower bound [Haviv & Langberg ‘11]

\[ c\sqrt{n} \leq L^*_{min} \leq \frac{n}{\log n} \]

- Coloring is the best upper bound we know on random graphs. Is it tight? OPEN
### Index Coding & Rank Minimization

<table>
<thead>
<tr>
<th>Wants: $X_1$</th>
<th>Has: $X_2$ $X_3$</th>
<th>$t_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wants:</strong> $X_4$</td>
<td><strong>Has:</strong> $X_1$</td>
<td>$X_1$ $X_2$ $X_3$ $X_4$</td>
</tr>
<tr>
<td><strong>Wants:</strong> $X_2$</td>
<td><strong>Has:</strong> $X_1$ $X_3$</td>
<td>$X_1+X_2$</td>
</tr>
<tr>
<td><strong>Wants:</strong> $X_3$</td>
<td><strong>Has:</strong> $X_2$ $X_4$</td>
<td>$X_1+X_4$ $X_4$</td>
</tr>
</tbody>
</table>

Matrix $M$

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$ $t_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$X_1+X_2$ $t_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$X_1+X_3+X_3$ $t_3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$X_1+X_4$ $X_4$ $t_4$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Linear case:** $L_{min}^* = \min r_k(M)$ [Bar-Yossef et al. '06]
- **Min rank introduced by Haemers in 79 to bound the Shannon graph capacity.**
- **Computing $L_{min}^*$ is NP hard.** [R. et al. '07] [Peeters '96]
- **Recent work on matrix completion for index coding** [Hassibi et al. ‘14]
Contributions

- Propose heuristics for solving index coding problem using rank minimization methods
- Compare to graph coloring solutions
- Matlab code for constructing
  1. Index codes (of course)
  2. Network codes for general networks
  3. Locally repairable codes
  4. Matroid representations
- Interesting case where an optimization problem results in an “actual” code
- Open: theoretical guarantees
Use Matrix Completion Methods to Construct Index Codes

- Min nuclear norm [Recht & Candes ‘09] does not work here
- Try alternative rank minimization methods [Fazel et al. 2001]

Theorem: [Alternating Projections (AP)]

If C and D are convex, then an alternating projection sequence between these 2 regions converges to a point in their intersection.
Algorithm APIndexCoding: Alternating projections method for index coding.

**Input:** Graph $G$ (or $G_d$)

**Output:** Completed matrix $M^*$ with low rank $r^*$

1. Set $r_k = \text{greedy coloring number of } \bar{G}$;
2. while $\exists M \in C'$ such that $\text{rank} M \leq r_k$ do
3. Randomly pick $M_0 \in C'$. Set $i = 0$ and $r_k = r_k - 1$;
4. repeat
5. $i = i + 1$;
6. /* Projection on $C'$ (resp. $C$) via eigenvalue decomposition (resp. SVD) */
7. Find the eigenvalue decomposition $M_{i-1} = U \Sigma V^T$, with $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$, $\sigma_1 \geq \cdots \geq \sigma_n$;
8. Set $\sigma_l = 0$ if $\sigma_l < 0$, $l = 1, \ldots, n$;
9. Compute $M_i = \sum_{j=1}^{r_k} \sigma_j u_j u_j^T$;
10. /* Projection on $D$ */ $M_{i+1} = M_i$ Set the diagonal entries of $M_{i+1}$ to 1's;
11. Change the $(a, b)^{th}$ position in $M_{i+1}$ to 0 if edge $(a, b)$ does not exist in $G$;
12. until $\|M_{i+1} - M_i\| \leq \epsilon$;
13. end
14. return $M^* = M_i$ and $r^* = r_k$. 

Index Coding via Alternating Proj on Random Undirected Graphs

$n=100$ users

Average Index Code Length vs. $p$

- No Coding
- Multicast
- Greedy Coloring
- Alternating Proj.
Improvement in Percentage

![Graphs showing improvement in percentage over various parameters and values of n.]
Concentration around the Average

The graph illustrates the concentration around the average with different broadcast lengths for various values of $p$.

Key features:
- The x-axis represents the broadcast length, ranging from 0 to 100.
- The y-axis represents the concentration, ranging from 0 to 0.8.
- Different lines represent different values of $p$:
  - $p=0.8$ (solid blue line)
  - $p=0.6$ (dashed green line)
  - $p=0.4$ (dash-dot red line)
  - $p=0.2$ (dotted blue line)

Legend:
- APIndexCoding (solid blue line)
- LDG (dashed green line)
- Greedy col. (dash-dot red line)
Running Time

s/graph

n: number of users (p=0.2)
Random Directed Graphs

![Graph showing expected values of a function with different algorithms]

- Greedy coloring
- LDG
- undirectedLDG
- SVDAP
- dirSYDAP
- AltPrj
- AltMin

The graph plots the expectation of a function against n, with various algorithms represented by different line colors and styles.
Which Method to use for Directed Graphs?
How close are these heuristics to the actual minimum?

For $n \leq 5$, linear index coding achieve capacity [Ong,’14]. Online list of optimal index coding rates [kim]

APIndex coding was able to achieve all these rates whenever they are integers.

**Fig. 9:** Average index code length obtained by using Greedy Coloring, LDG and APIndexCoding for random 3-colorable graphs when $p = 0.5$. 
“Application” to Network Coding & Storage & Matroids
Goal

Wants: \( X_2 \)

Output: Network Code

\[
\begin{array}{c}
X_1 \\
\uparrow \\
S_1 \\
\downarrow e_1 \\
\bigtriangleup \\
\downarrow e_5 \\
\bigtriangleup \\
\downarrow e_6 \\
\bigtriangleup \\
\downarrow e_7 \\
\bigtriangleup \\
\downarrow e_4 \\
\bigtriangleup \\
\downarrow e_3 \\
\bigtriangleup \\
\downarrow e_2 \\
\bigtriangleup \\
X_2 \\
S_2 \\
\downarrow t_2 \\
\bigtriangleup \\
\downarrow t_1 \\
\bigtriangleup \\
X_1
\end{array}
\]

Command Window

\[
\begin{align*}
\text{>> } & \text{Butterfly\_Network} = \begin{bmatrix} 1 & 5 \\
1 & 3 \\
2 & 3 \\
2 & 6 \\
3 & 4 \\
4 & 5 \\
4 & 6 \\
\end{bmatrix}; \\
\text{>> } & \text{Demand} = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 1 \\
\end{bmatrix}; \\
\text{>> } & \text{NC} = \text{FindNetworkCode(Butterfly\_Network,Demand)}
\end{align*}
\]
An index code of length $L$ that satisfies all the users

A network code that satisfies all the terminals
**Theorem:** [R,Sprintson, Georghiades’08] [Effros,R,Langberg ISIT’13]

For any network coding problem, one can construct an index coding problem and an integer $L$ such that given any network code, one can efficiently construct an index code of length $L$, and vice versa. (same block length, same error probability).
Butterfly network

All terminals in the index coding problem can decode

Any linear network code gives an index code of length $L=7$
Butterfly Network

\[ X_1 \xrightarrow{s_1} e_2 \xrightarrow{e_3} X_2 \]

\[ e_1 \xrightarrow{e_5} e_4 \xrightarrow{e_6} e_7 \]

\[ t_2 \xrightarrow{e_2} e_3 \xrightarrow{e_4} X_1 \]

\[ \text{Wants: } X_2 \]

Command Window

\[
\begin{align*}
\text{Butterfly\_Network} & = [1 \ 5; 1 \ 3; 2 \ 3; 2 \ 6; 3 \ 4; 4 \ 5; 4 \ 6]; \\
\text{Demand} & = [0 \ 0 \ 0 \ 0 \ 2 \ 1]; \\
\text{FindNetworkCode}(\text{Butterfly\_Network}, \text{Demand})
\end{align*}
\]

Equivalent Index Coding Problem

\[
\begin{align*}
M[i,j] & \text{ is the message on edge}[i,j]; \\
D[k] & \text{ is the decoding message on node } k; \\
M[1,3] & = (-0.754565) \times X_1; \\
M[1,5] & = (1.245211) \times X_1; \\
M[2,3] & = (0.471860) \times X_2; \\
M[2,6] & = (0.867619) \times X_2; \\
M[3,4] & = (0.936511) \times M[1,3] + (1.296273) \times M[2,3]; \\
M[4,5] & = (1.812115) \times M[3,4]; \\
M[4,6] & = (-0.976809) \times M[3,4]; \\
D[5] & = (0.932065) \times M[1,5] + (0.909083) \times M[4,5]; \\
D[6] & = (0.990609) \times M[2,6] + (1.438965) \times M[4,6];
\end{align*}
\]
\[
X_1 = 0.8126m_4 + 0.6722m_7 \\
X_2 = 1.5346m_1 + 1.2585m_6 \\
m_2 = 0.9391X_1 \\
m_3 = 1.345X_2 \\
m_4 = 1.038X_2 \\
m_5 = 1.413m_2 - 0.8321m_3 \\
m_6 = -0.71m_5 \\
m_7 = 1.1211m_5 \\
m_1 = 0.7726 \\
m_5 = 0.71 \\
m_7 = 1.1211 \\
\]

Command Window:

```matlab
>> Butterfly_Network=[1 5;1 3;2 3;2 6;3 4;4 5;4 6];
>> Demand=[0 0 0 0 2 1];
>> FindNetworkCode(Butterfly_Network,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
e1 M[1,3] = (-0.754565)*X1 

e2 M[1,5] = (1.245211)*X1 

e3 M[2,3] = (0.471860)*X2 

e4 M[2,6] = (0.867619)*X2 

e5 M[3,4] = (0.936511)*M[1,3] + (1.296273)*M[2,3] 

e6 M[4,5] = (1.812115)*M[3,4] 

e7 M[4,6] = (-0.976809)*M[3,4] 

t1 D[5] = (0.932065)*M[1,5] + (0.900983)*M[4,5] 

t2 D[6] = (0.990609)*M[2,6] + (1.438965)*M[4,6];
```
Example 2

Wants: $X_3$

Wants: $X_2$

Wants: $X_1$

$e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}$

$t_1$ $s_1$ $X_1$

$t_2$ $s_2$ $X_2$

$t_3$ $s_3$ $X_3$

$e_{14}$ $e_{15}$ $e_{16}$

$e_{17}$ $e_{18}$ $e_{19}$

$D[k]$ is the decoding message on node $k$;

$M[i,j]$ is the message on edge $[i,j]$;

- $M[1,4] = (0.586472)X_1$
- $M[1,6] = (2.266747)X_1$
- $M[1,7] = (-0.063189)X_1$
- $M[2,4] = (0.509654)X_1$
- $M[2,6] = (2.666987)X_2$
- $M[2,8] = (0.911907)X_2$
- $M[3,4] = (1.119271)X_3$
- $M[3,7] = (0.655653)X_3$
- $M[3,8] = (1.513620)X_3$
- $M[5,12] = (0.846141)M[4,5]$
- $M[5,13] = (0.978085)M[4,5]$
- $M[5,14] = (1.593188)M[4,5]$
- $M[6,9] = (0.959276)M[1,6] + (1.882469)M[2,6]$
- $M[7,10] = (1.269984)M[1,7] + (-0.506044)M[3,7]$
- $M[9,12] = (-0.367560)M[6,9]$
- $M[10,13] = (2.090403)M[7,10]$
- $D[12] = (0.863644)M[5,12] + (0.303506)M[9,12]$
Solution of example 2

```matlab
>> Example2_Network=[1 6;1 7;1 4;2 6;2 4;2 8;3 4;3 7;3 8;4 5;5 12;5 13;5 14;6 9;7 10;8 11;9 12;10 13;11 14];
>> Demand=zeros(1,11) 3 2 1;
>> FindNetworkCode(Example2_Network,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,4] = (0.586472)*X1;
M[1,6] = (2.266747)*X1;
M[1,7] = (-0.063189)*X1;
M[2,4] = (0.589654)*X2;
M[2,6] = (2.66987)*X2;
M[2,8] = (0.911907)*X2;
M[3,4] = (1.19271)*X3;
M[3,7] = (0.655653)*X3;
M[3,9] = (1.513620)*X3;
M[4,5] = (0.565311)*M[1,4] + (1.295912)*M[2,4] + (1.223358)*M[3,4];
M[5,12] = (0.846141)*M[4,5];
M[5,13] = (0.978085)*M[4,5];
M[5,14] = (1.593188)*M[4,5];
M[6,9] = (0.959276)*M[1,6] + (1.882469)*M[2,6];
M[7,10] = (1.269984)*M[1,7] + (-0.506044)*M[3,7];
M[8,11] = (1.808865)*M[2,8] + (1.948409)*M[3,8];
M[9,12] = (-0.367560)*M[6,9];
M[10,13] = (2.090403)*M[7,10];
M[11,14] = (2.417460)*M[8,11];
D[12] = (0.863644)*M[5,12] + (0.303506)*M[9,12];
D[13] = (1.292302)*M[5,13] + (2.521777)*M[10,13];
D[14] = (1.896364)*M[5,14] + (-0.579582)*M[11,14];
```
Non – linear code

[Dougherty, Freiling, Zeger, ‘05]

Command Window

>> Network2=[1 4;2 4;3 5;4 6;5 7;6 8;6 9;7 8;7 14;3 9;8 10;9 11;10 12;10 13;11 13;11 14;1 12];
>> Demand=[0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 1];
>> NC=FindNetworkCode(Network2,Demand)
Cannot find scalar linear network code.

NC =
[
]

Example

S. Kamath et al., Generalized Network Sharing Outer Bound and the Two-Unicast Problem, 2011

---

```
Command Window

>> Network_fig1=[[1 3; 2 3; 2 5; 3 4; 4 4 7; 4 5; 5 6; 6 6 7; 6 8];
>> Demand=zeros(1,6) 2 1;
>> FindNetworkCode(Network_fig1,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,3] = (-0.882440)*X1;
M[2,3] = (0.939841)*X2;
M[2,5] = (0.383182)*X2;
M[3,4] = (2.289987)*M[1,3] + (1.721564)*M[2,3];
M[4,5] = (0.615945)*M[3,4];
M[4,7] = (2.291414)*M[3,4];
M[5,6] = (1.606550)*M[2,5] + (-0.611804)*M[4,5];
M[6,7] = (1.246499)*M[5,6];
M[6,8] = (1.835592)*M[5,6];
D[7] = (0.268142)*M[4,7] + (1.299558)*M[6,7];
D[8] = (0.705990)*M[6,8];
```
Examples

```
>> Network_fig2a=[1 3;2 3;2 5;2 7;3 4;4 10;4 5;5 6;6 7;6 10;7 8;8 9;8 10];
>> Demand=zeros(1,8)*1 2;
>> FindNetworkCode(Network_fig2a,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,3] = (0.357853)*X1;
M[2,3] = (1.160702)*X2;
M[2,5] = (0.873565)*X2;
M[2,7] = (-0.145219)*X2;
M[3,4] = (-0.313999)*M[1,3] + (0.817991)*M[2,3];
M[4,5] = (-0.626207)*M[3,4];
M[4,10] = (0.880154)*M[3,4];
M[5,6] = (0.854540)*M[2,5] + (1.230356)*M[4,5];
M[6,7] = (1.472249)*M[5,6];
M[6,10] = (1.092611)*M[5,6];
M[7,8] = (0.304585)*M[2,7] + (1.737330)*M[6,7];
M[8,9] = (2.388991)*M[7,8];
M[8,10] = (0.565171)*M[7,8];
D[9] = (1.855486)*M[8,9];
D[10] = (1.174221)*M[4,10] + (0.997828)*M[6,10] + (0.148880)*M[8,10];
```
Constructing Linear Repairable Codes* is equivalent to constructing linear index codes

- [Mazumdar ‘14],[Shanmugam, Dimakis’14]

Command Window

```
>> Pentagon=[1 2;2 3;3 4;4 5;1 5];
>> [StorageCode,FileSize]=FindLRC(Pentagon)

StorageCode =

       0         0         0         0         0         0
    0.5859       0   0.4915   0.0000   0.0000   0.4806
    0         0   0.8669   0.7300   0.0000   0.0000
    0         0         0   0.5639   0.0000   0.4618
  1.0807         0         0   0.6124   0.0000   0.0000

FileSize =
```
Locally Repairable Code

Command Window

```matlab
Pentagon=[1 2; 2 3; 3 4; 4 5; 1 5];
[StorageCode, FileSize] = FindLRC(Pentagon)
```

```
StorageCode =
0 0.4708 0 0 0.4806
0.5859 0 0.4915 0 0
0 0.8669 0 0.7300 0
0 0 0.5639 0 0.4618
1.0807 0 0 0.6124 0
```

FileSize =
```
2
```

\[
X_3 = -0.8669X_2 + 0.73X_4 \quad X_4 = 0.5639X_3 + 0.4618X_5
\]

\[
X_2 = 0.5859X_1 - 0.4915X_3 \quad X_5 = 1.0807X_1 + 0.6124X_4
\]

\[
X_1 = 0.4708X_2 + 0.4806X_5
\]

\[
X_3 = \alpha_2 \quad X_4 = 0.6958\alpha_1 + 0.7861\alpha_2
\]

\[
X_2 = 0.5859\alpha_1 - 0.4915\alpha_2 \quad X_5 = 1.5068\alpha_1 + 0.4812\alpha_2
\]

\[
X_1 = \alpha_1
\]
Another Example

Command Window

```
>> DSS=[1 2;1 3;1 4;1 5;2 3;3 4;4 5];
>> [StorageCode,FileSize]=FindLRC(DSS)
```

StorageCode =

```
   0   0.7637   0.8404   0.4520   0.5938
 1.0840         0   1.1005         0   0
 0.9850   0.9087         0   0.0000   0
 0.3809         0         0   0.0000   1.3135
 0.2900         0         0   0.7613         0
```

FileSize =

3

\[
X_3 = 0.985X_1 - 0.9087X_2 \quad X_4 = 0.3809X_1 - 1.3135X_5
\]

\[
X_2 = 1.084X_1 - 1.1005X_3
\]

\[
X_1 = 0.7637X_2 + 0.8404X_3 + 0.452X_4 + 0.5938X_5
\]

\[
X_3 = \alpha_2
\]

\[
X_4 = \alpha_3
\]

\[
X_2 = 1.084\alpha_1 - 1.1005\alpha_2
\]

\[
X_5 = 0.29\alpha_1 - 0.7613\alpha_3
\]

\[
X_1 = \alpha_1
\]
Index coding is NP hard. But, this is not the end of the story.
Proposed the use of different rank minimizations methods for constructing index codes
Index coding is connected to many other interesting topic in the literature
Building a matlab library to
  - Construct network codes
  - Codes with locality for distributed storage
  - Matroid representations
Open questions:
Provide theoretical guarantees on the performance of these algorithms
How to go from the reals to finite fields?
Salim El Rouayheb – Software

Index Coding via Rank Minimization

Matlab code

The Matlab code above implements various rank minimization methods (Alternating Projections, Directional Alternating Projections, etc.) to construct near-optimal index codes.

Full Paper available on Arxiv.
Thank You!

QUESTIONS?
Lemma 3: Let $\mathbf{X} = [X_1, X_2, \ldots, X_n]^T$ be the message vector at the transmitter. Assume that the index code given by matrix $M^*$ is used and let $\hat{\mathbf{X}} = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_n]^T$ be the messages decoded by the users. Then,

$$
\| \mathbf{X} - \hat{\mathbf{X}} \| \leq \epsilon X_{\text{max}} \sqrt{n}. \quad (5)
$$

\[
\begin{align*}
\| \mathbf{X} - \hat{\mathbf{X}} \| &= \| \mathbf{X} - M^* A^\dagger A \mathbf{X} - M^* \circ \Phi \mathbf{X} \| \\
&= \| \mathbf{X} - M^* \mathbf{X} - M^* \circ \Phi \mathbf{X} \| \\
&= \| (I + M^* \circ \Phi - M^*) \mathbf{X} \| \\
&= \| (M_D - M^*) \mathbf{X} \| \\
&\leq \| M_D - M^* \| \| \mathbf{X} \| \\
&= \| U \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_{n-r^*} \end{bmatrix} V^T \| X_{\text{max}} \sqrt{n} \\
&\leq \sigma_{r^*+1} X_{\text{max}} \sqrt{n} \\
&\leq \epsilon X_{\text{max}} \sqrt{n}.
\end{align*}
\]
Fig. 12: Average decoding error $\|X - \hat{X}\|$ in APIndexcoding on random undirected graphs when $p = 0.2$, $\epsilon = 0.001$ and $X_i \in [-10, 10]$ ($X_{\text{max}} = 10$).