Network Coding and Index Coding via Rank Minimization

Salim El Rouayheb

IIT, Chicago

Joint work with Xiao Huang

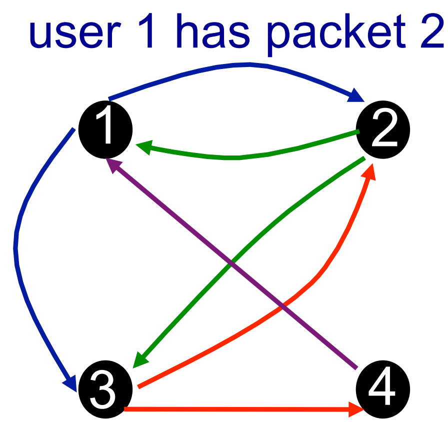
Index Coding

Trans-mission #	Index code 1	Index code2
1	X_1	X_1+X_2
2	X_2	X_3
3	X_3	X_4
4	X_4	

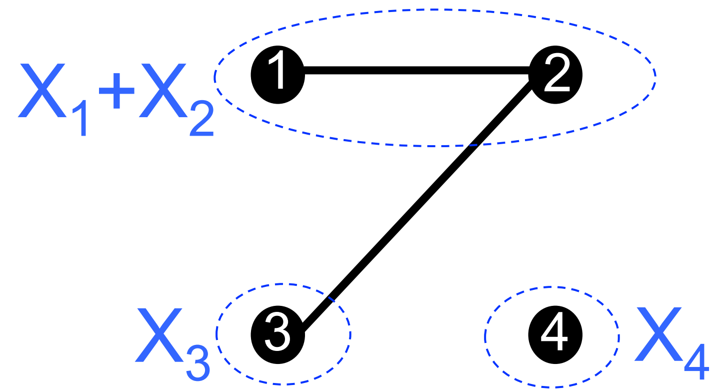
L=4 L=3

Informed-source coding-on-demand [Birk & Kol infocom'98]

Index Coding & Graph Coloring



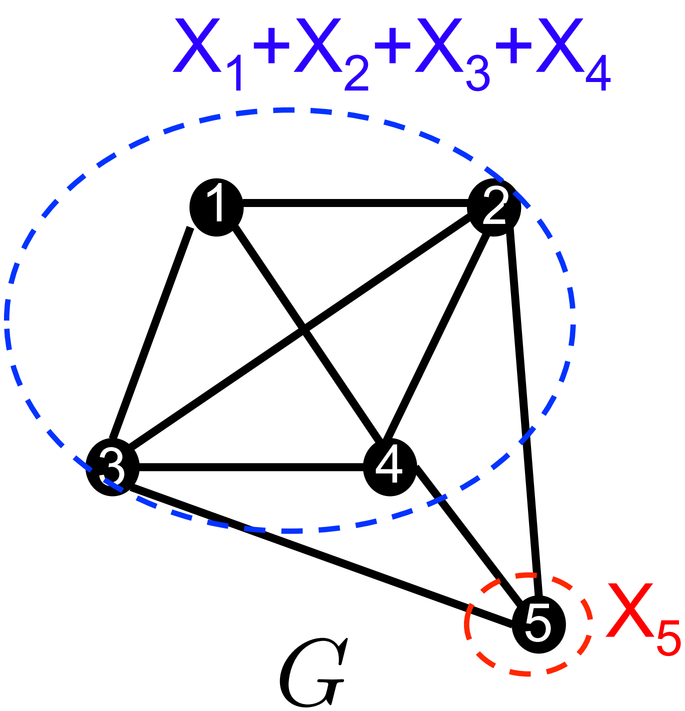
Side info graph G_d

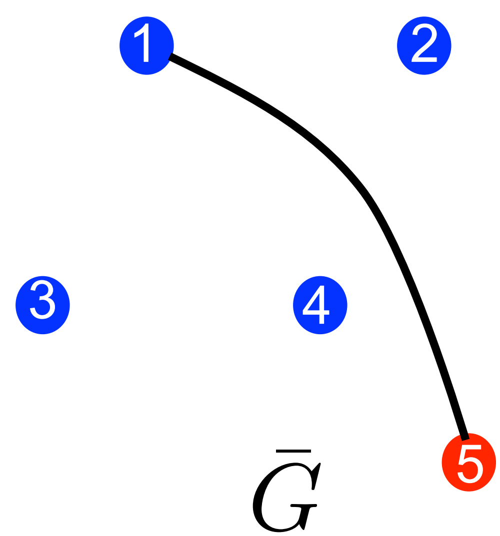


Clique cover of G

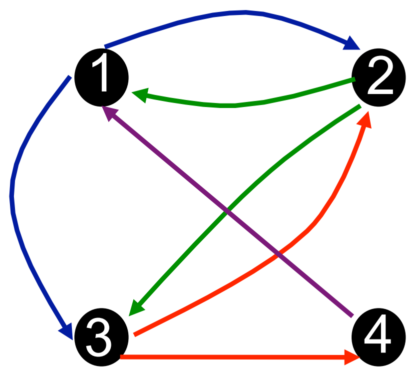
=

Chromatic nbr of $\bar{G}$

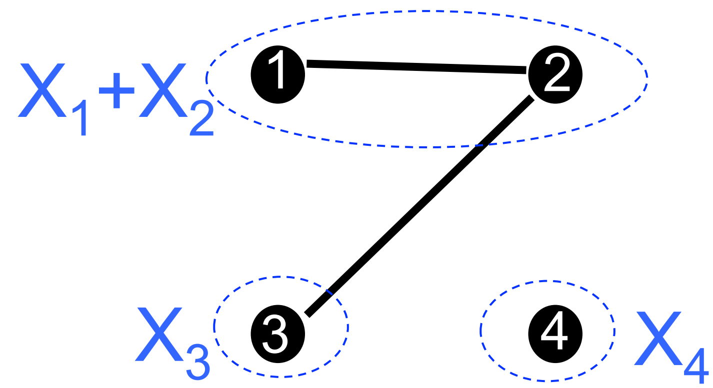




Index Coding & Graph Coloring



Side info graph G_d



Chromatic nbr of $\bar{G}$

Independence nbr

$$\alpha(G_d) \leq c(G_d) \leq L^*_{min} \leq \chi_f(\bar{G}) \leq \chi(\bar{G})$$

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]

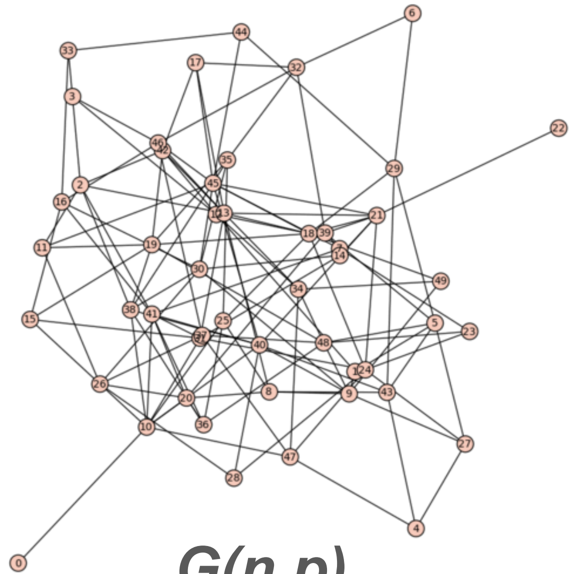
[Arbabjolfaei et al., '13]

…

Index Coding on Erdős-Rényi Graphs

Independence nbr   Chromatic nbr

$$\alpha(G) \leq L^*_{min} \leq \chi(\bar{G})$$



G(n,p)

- 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

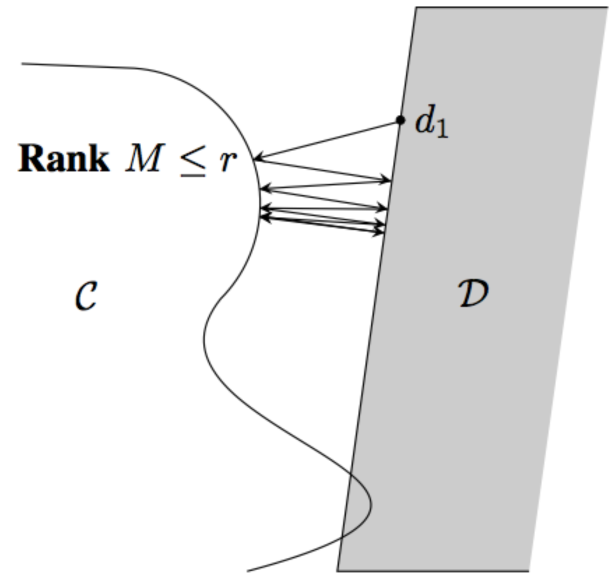
	X_1	X_2	X_3	X_4
X_1 t_1	1	1	1	0
X_1+X_2 t_2	1	1	1	0
$X_1+X_2+X_3$ t_3	0	1	1	1
X_1+X_4 X_4 t_4	1	0	0	1

Matrix M

- Linear case: $L^*_{min} = \min rk(M)$ [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

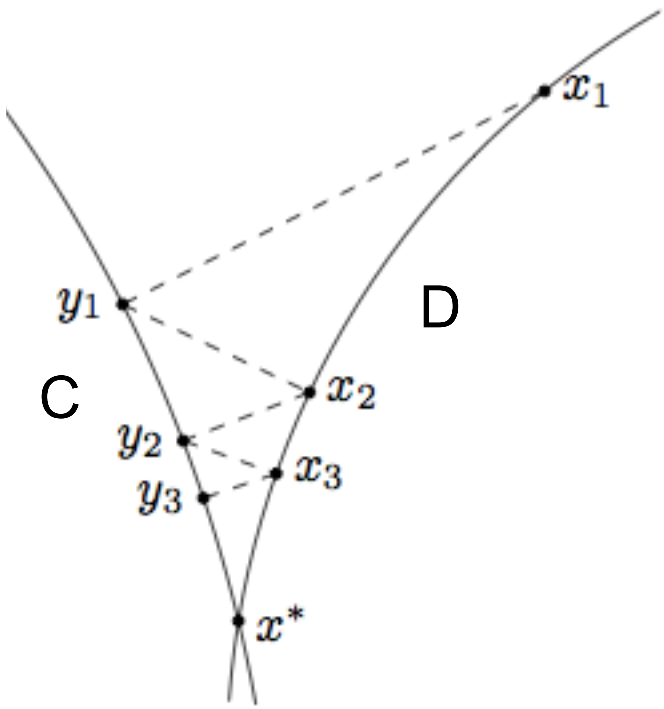


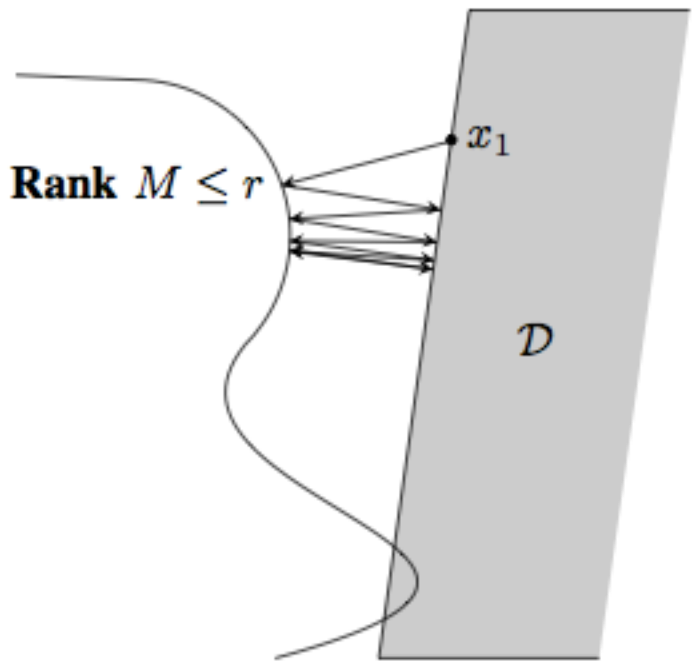
Index coding via Alternating Projections

- 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]





Index coding via AP

Two problems:

1) Regions not convex
2) Optimization over the reals

Network codes over the reals [Shwartz & Medard '14], Jaggi et al. '08]

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^*

Set r_k = greedy coloring number of $\bar{G}$;

while $\exists M \in \mathcal{C}'$ *such that rank* $M \leq r_k$ **do**

  Randomly pick $M_0 \in \mathcal{C}'$. Set $i = 0$ and $r_k = r_k - 1$;

  repeat

    $i = i + 1$;

    `/* Projection on` $\mathcal{C}'$ `(resp.` $\mathcal{C}$`) via eigenvalue decomposition (resp. SVD) */`

    Find the eigenvalue decomposition $M_{i-1} = U\Sigma V^T$, with $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$, $\sigma_1 \geq \cdots \geq \sigma_n$;

    Set $\sigma_l = 0$ if $\sigma_l < 0$, $l = 1, \ldots, n$;

    Compute $M_i = \sum_{j=1}^{r_k} \sigma_j u_j v_j^T$;

    `/* Projection on` $\mathcal{D}$ `*/`

    $M_{i+1} = M_i$ Set the diagonal entries of M_{i+1} to 1's;

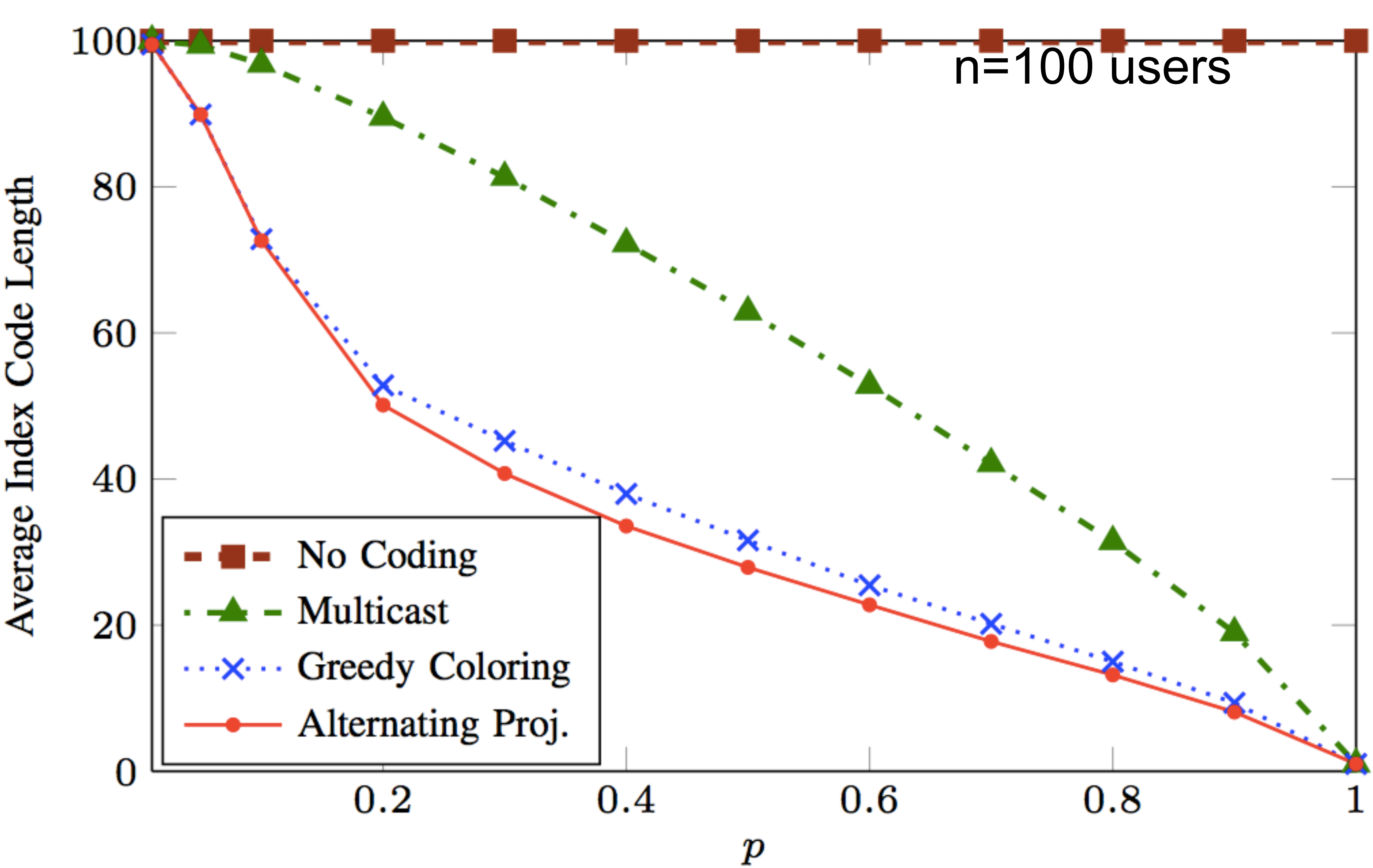
    Change the $(a, b)^{th}$ position in M_{i+1} to 0 if edge (a, b) does not exist in G;

  until $\|M_{i+1} - M_i\| \leq \epsilon$;

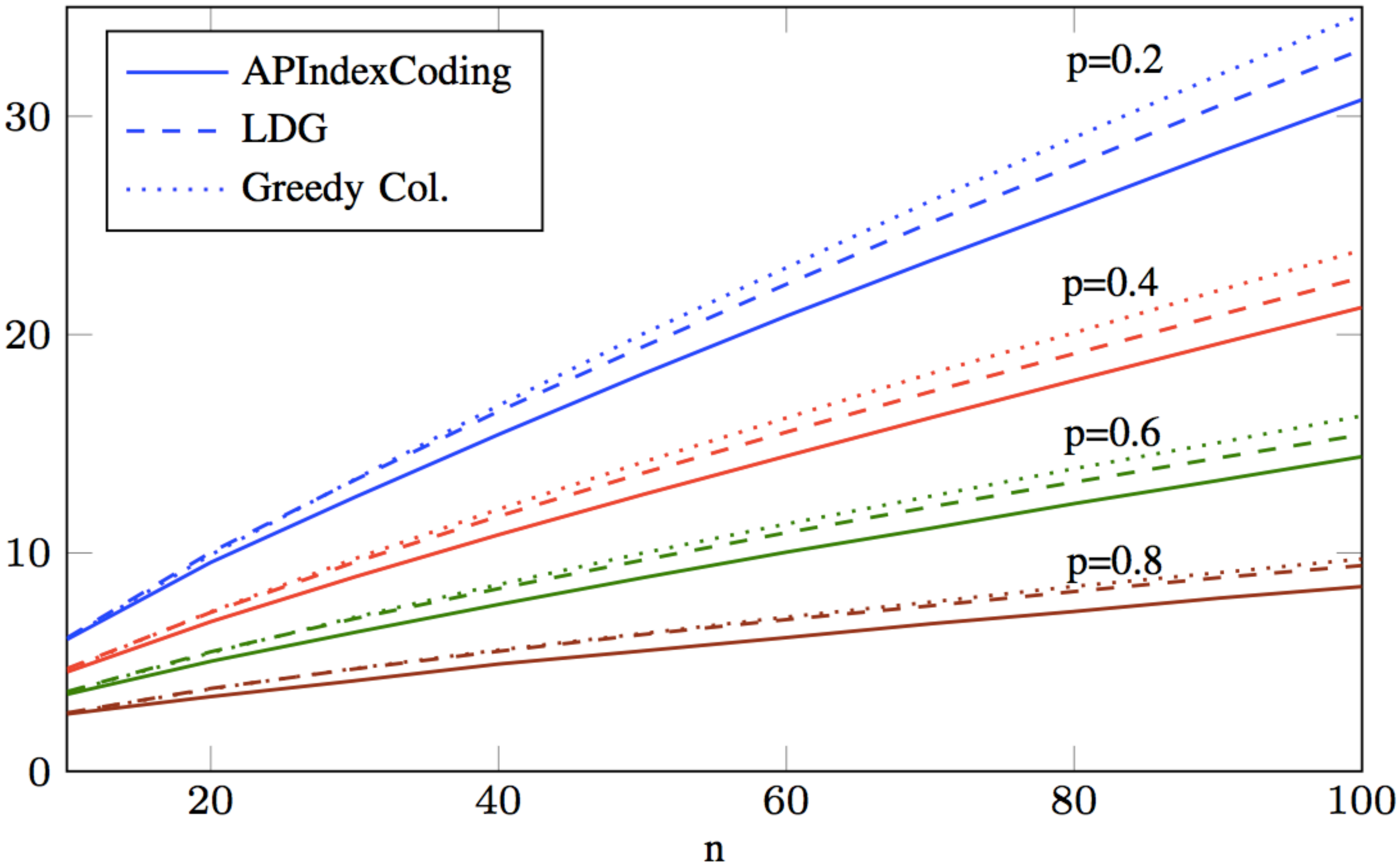
end

return $M^* = M_i$ and $r^* = r_k$.

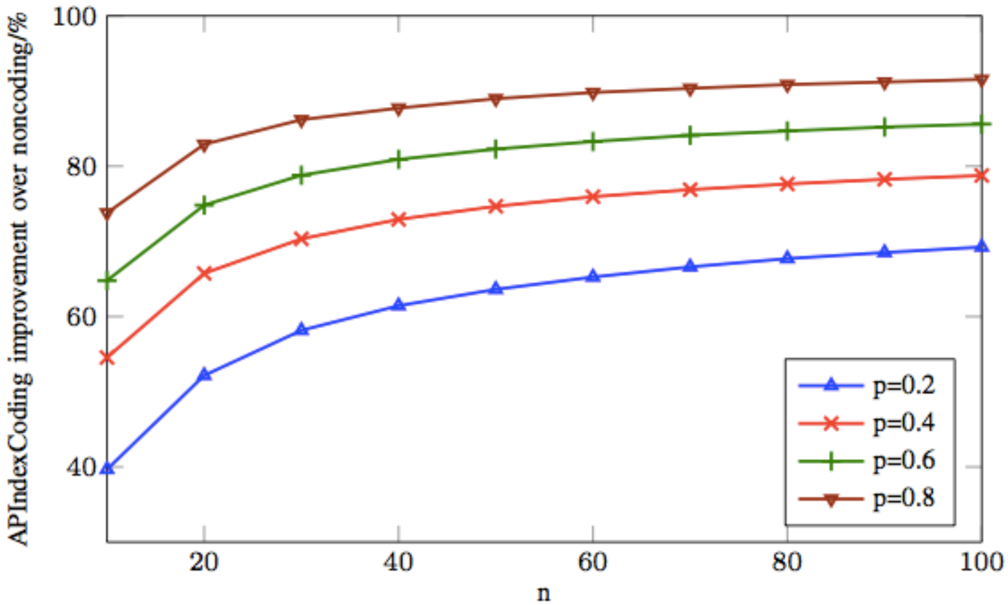
Index Coding via Alternating Proj on Random Undirected Graphs

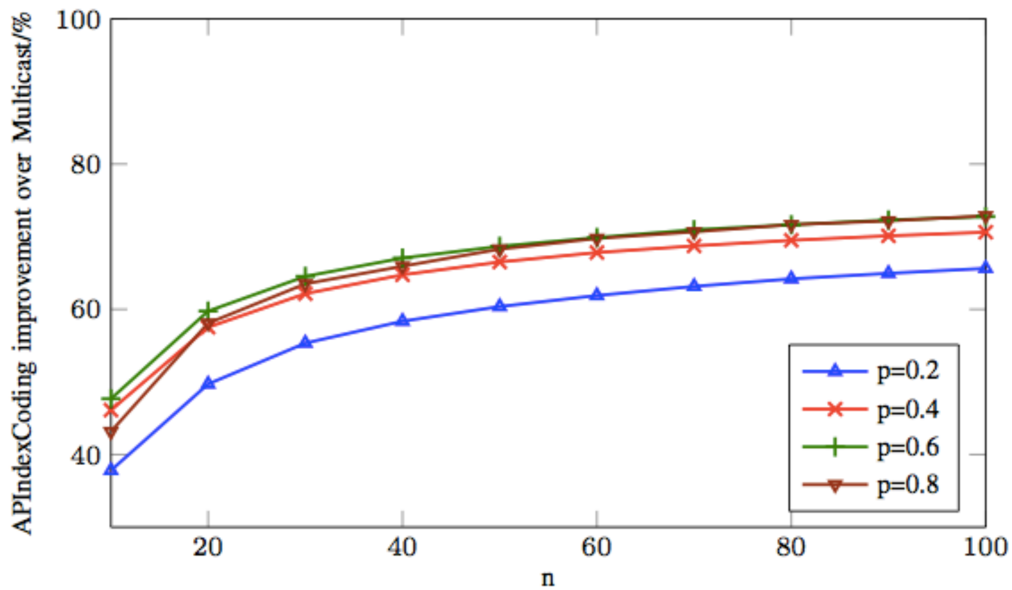


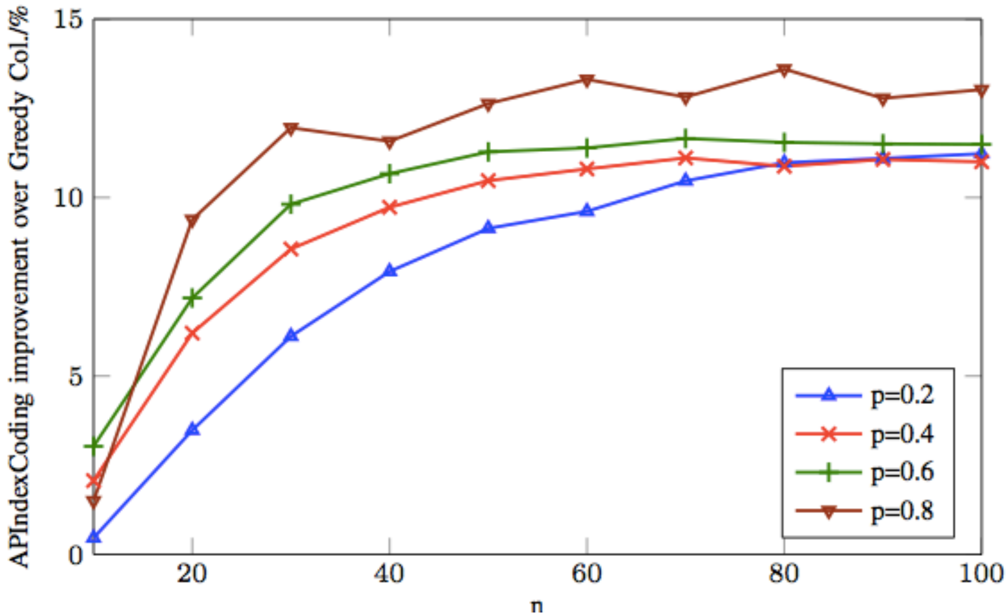
Performance with Increasing Number of Users

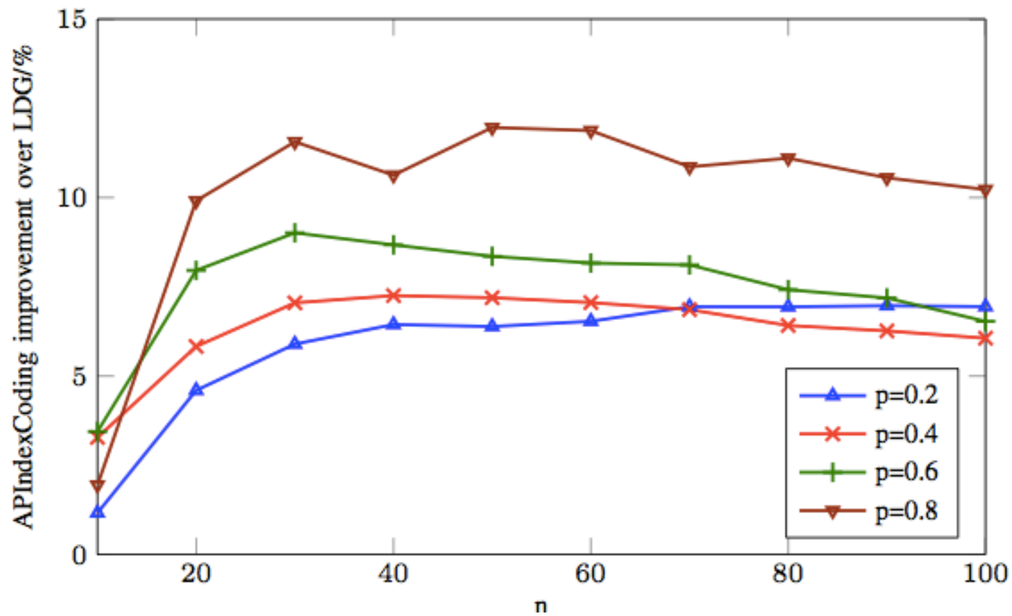


Improvement in Percentage

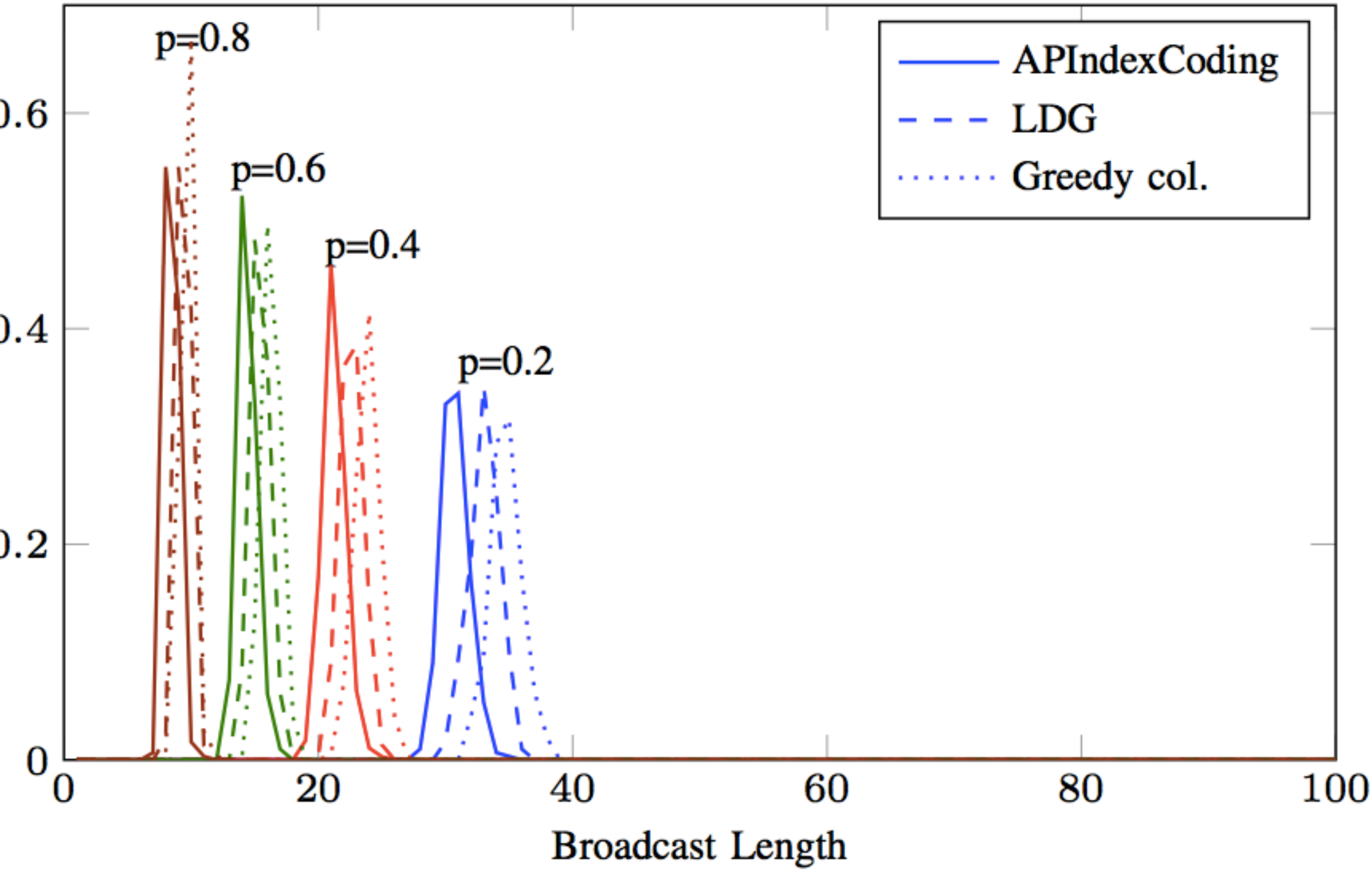




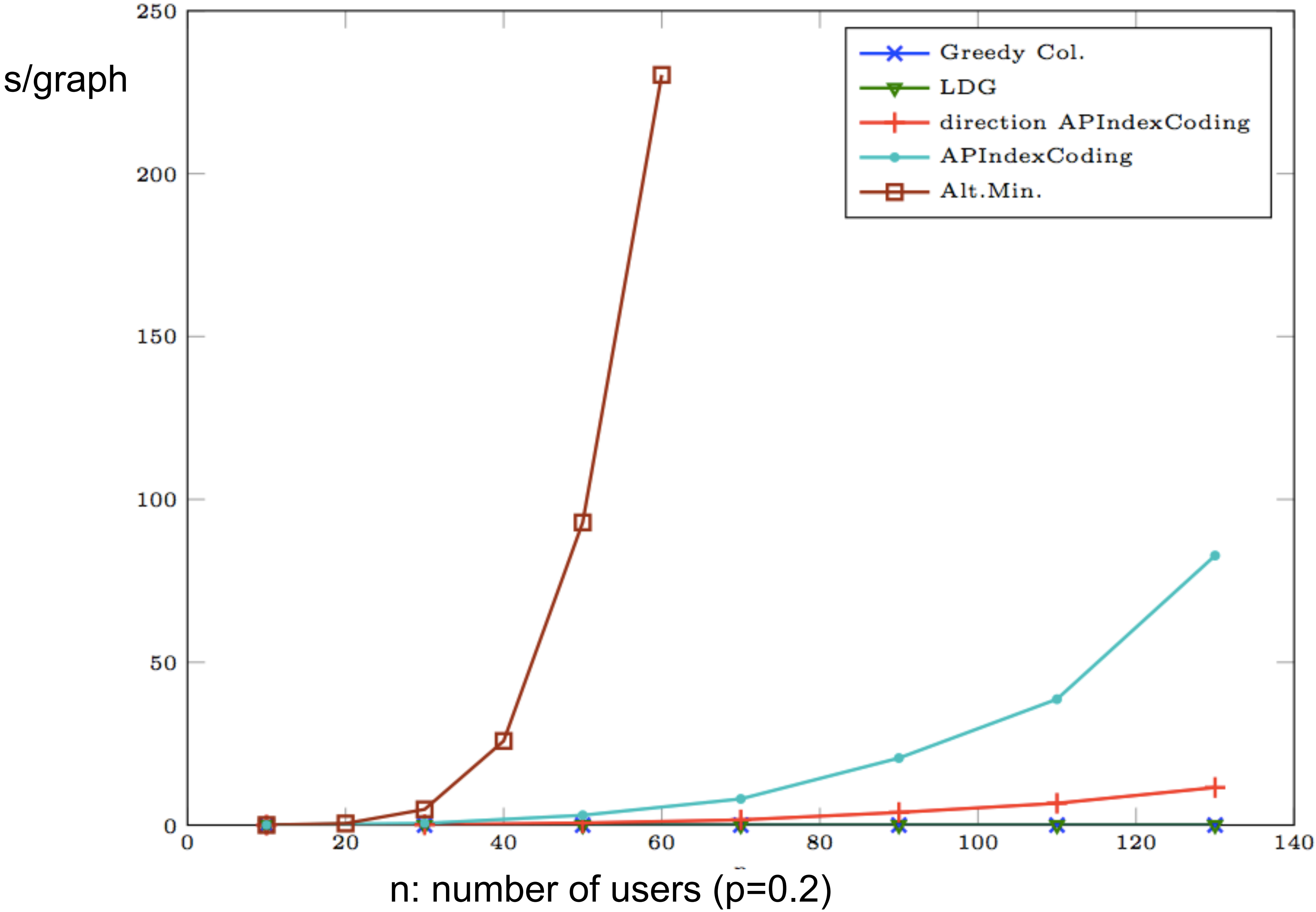




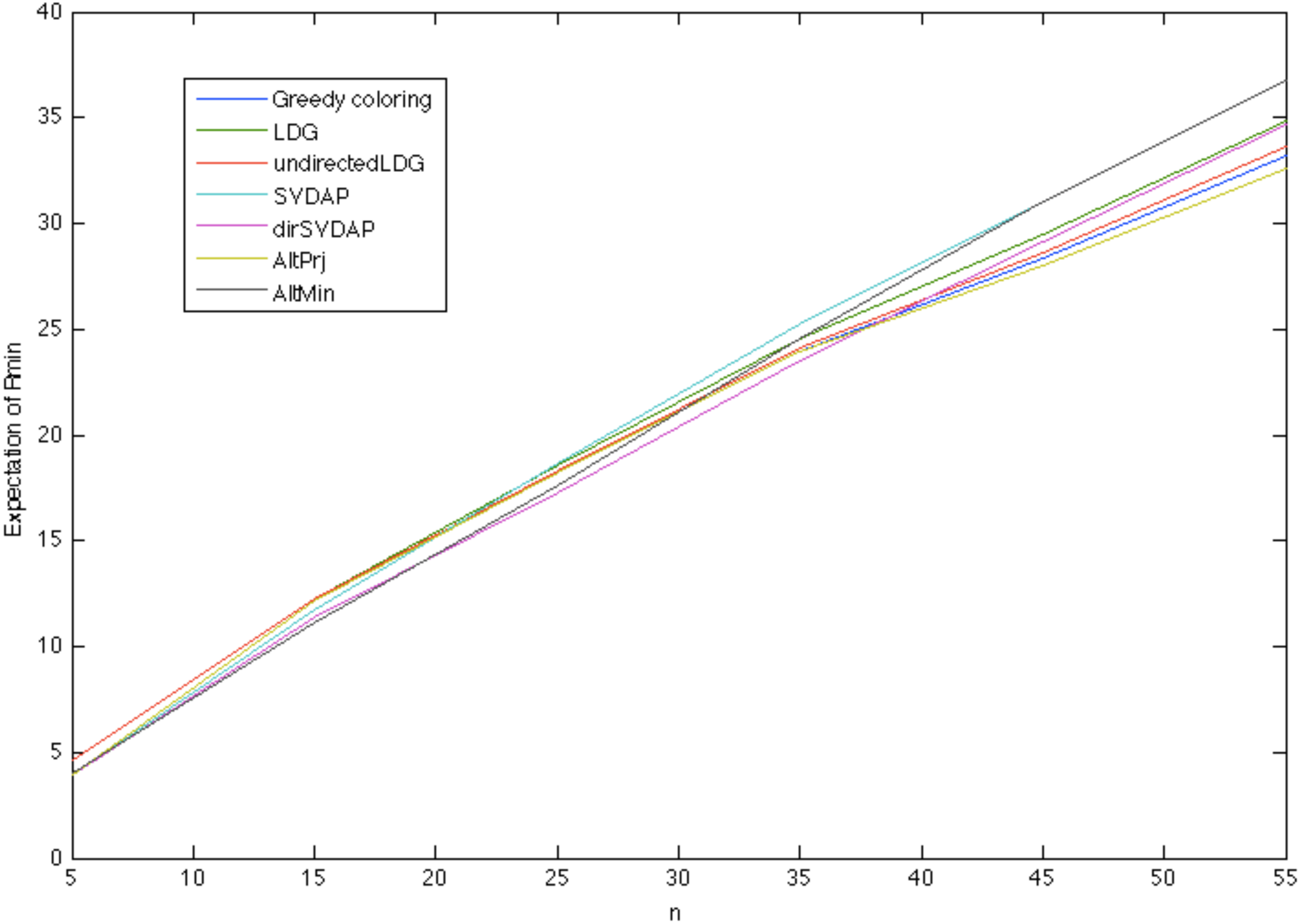
Concentration around the Average



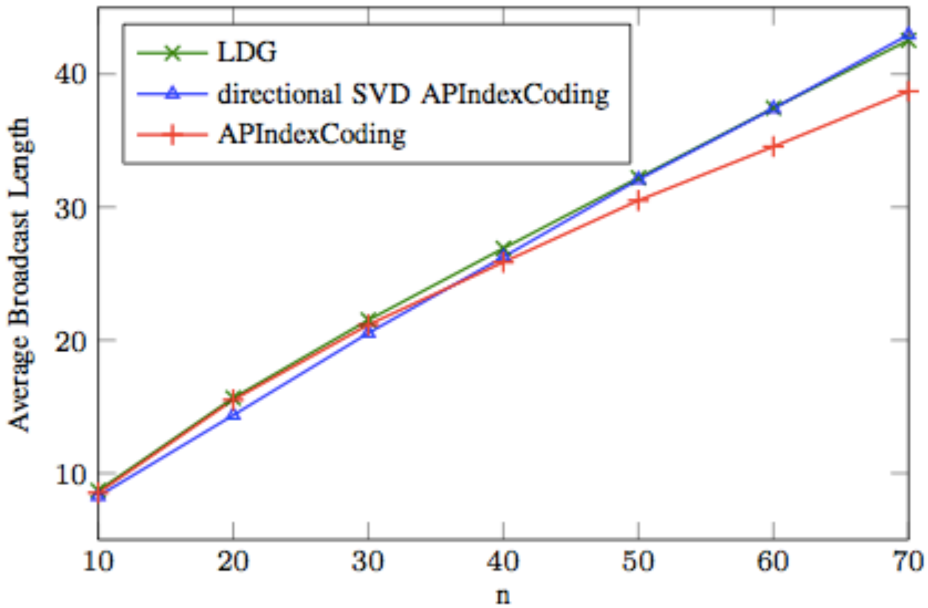
Running Time

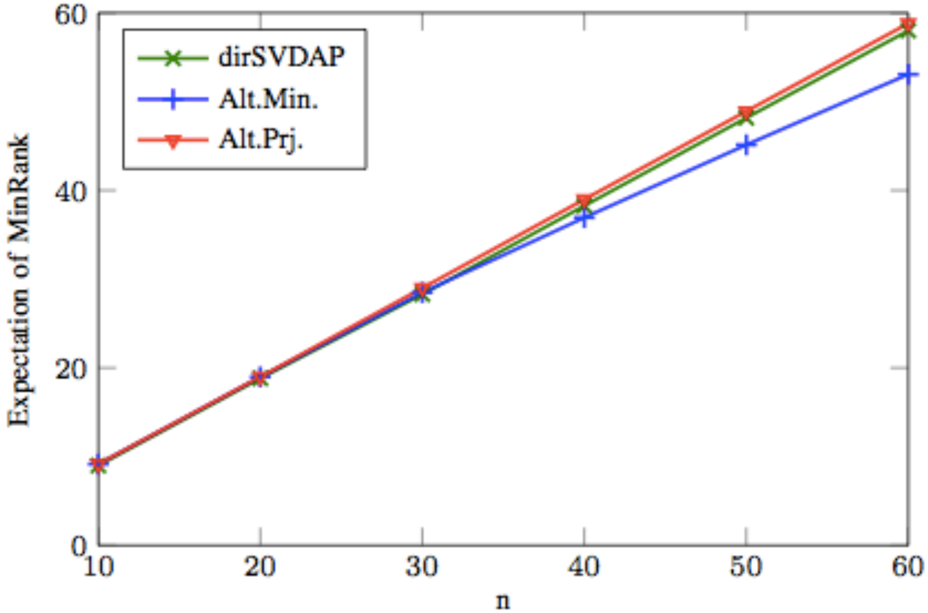


Random Directed Graphs



Which Method to use for Directed Graphs?





How close are these heuristics to the actual minimum

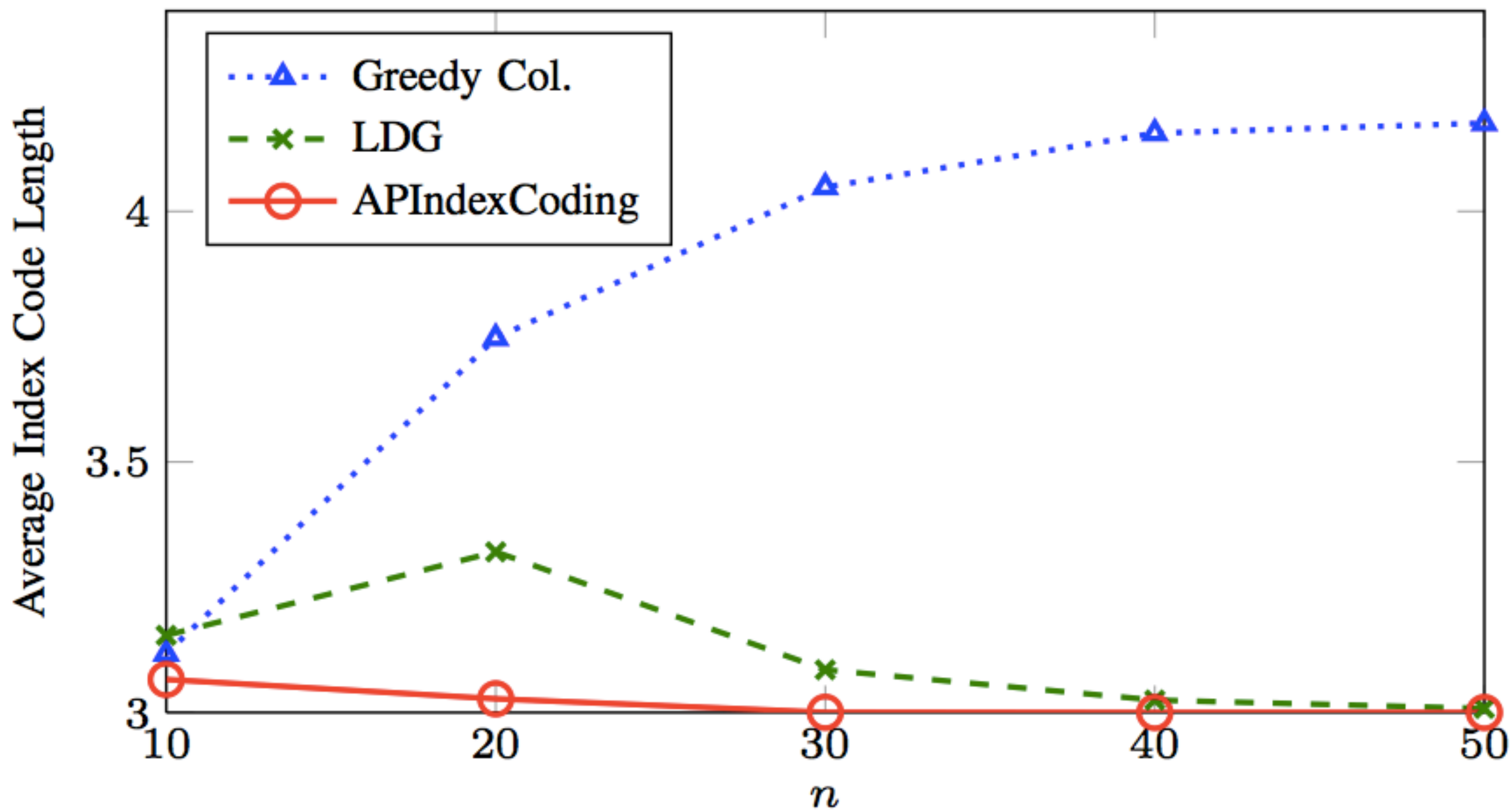


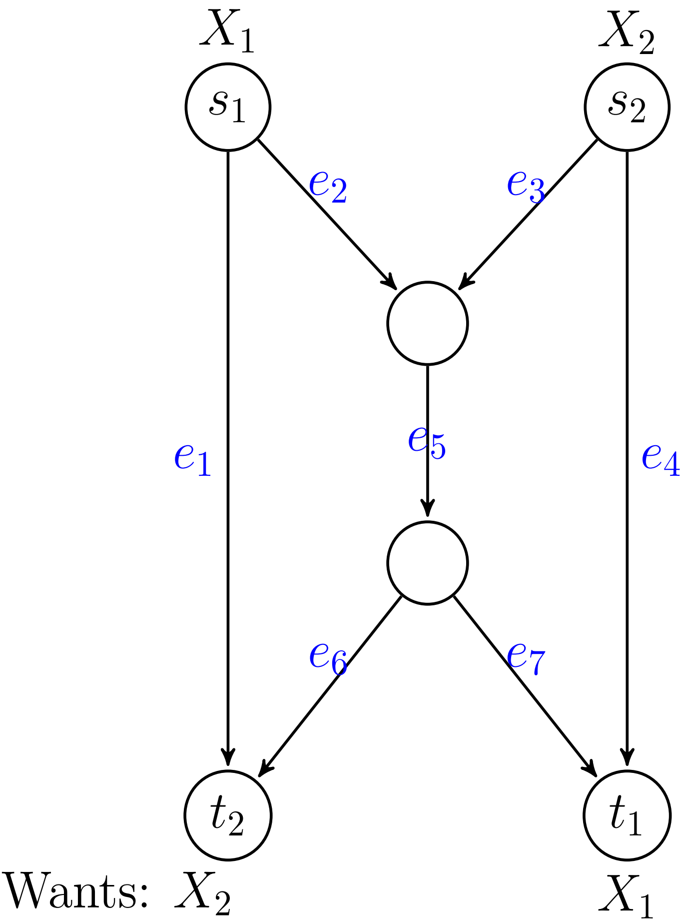
Fig. 9: Average index code length obtained by using Greedy Coloring, LDG and APIndexCoding for random 3-colorable graphs when $p = 0.5$.

- For n≤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

Next

“Application” to
Network Coding
& Storage
& Matroids

Goal

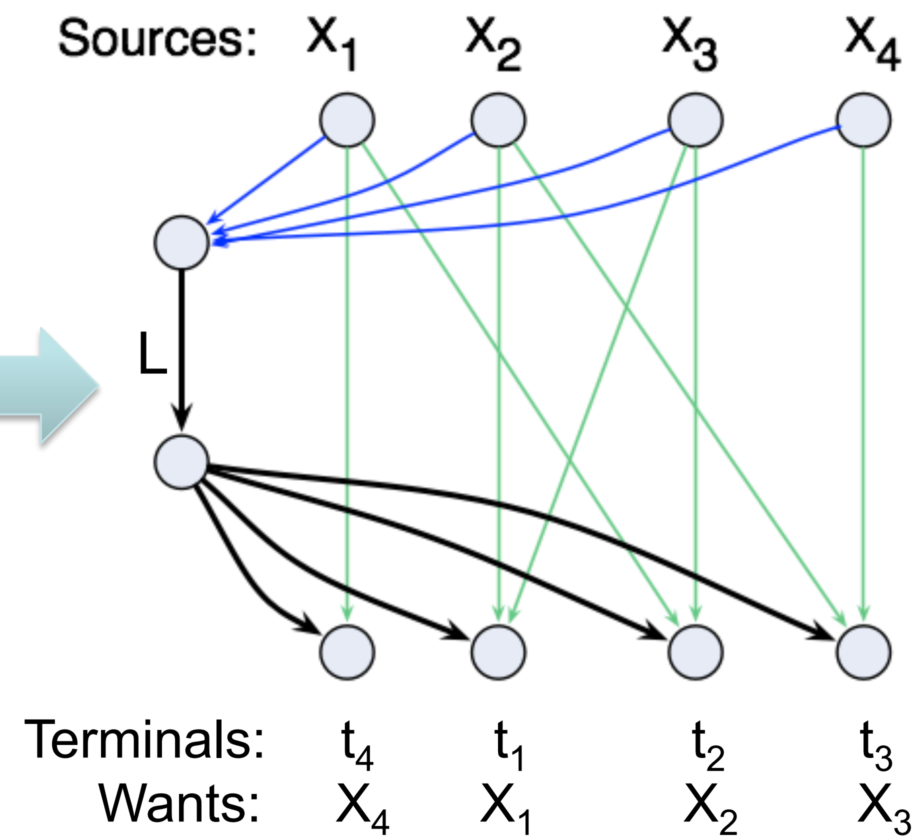


Command Window

```
>> Butterfly_Network=[1 5 ;1 3;2 3;2 6;3 4;4 5;4 6];
>> Demand=[0 0 0 0 2 1];
>> NC=FindNetworkCode(Butterfly_Network,Demand)
```

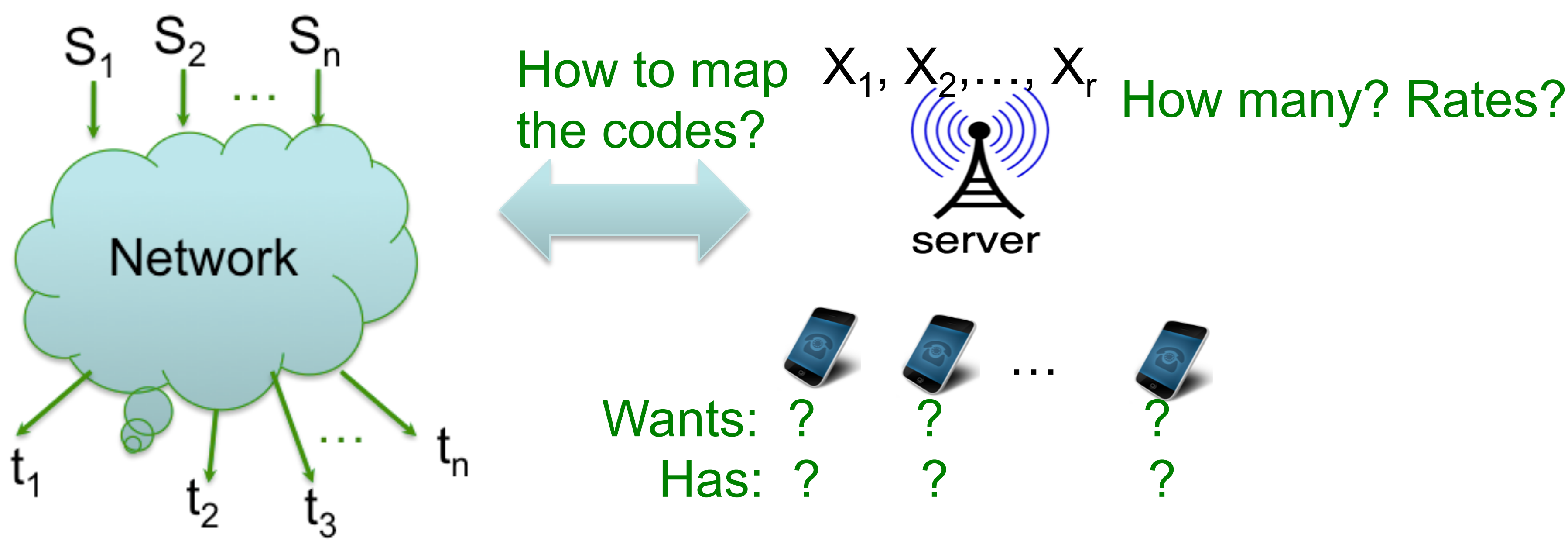
Output: Network Code

Equivalence to Network Coding



An index code of length L that satisfies all the users ⟺ **A network code that satisfies all the terminals**

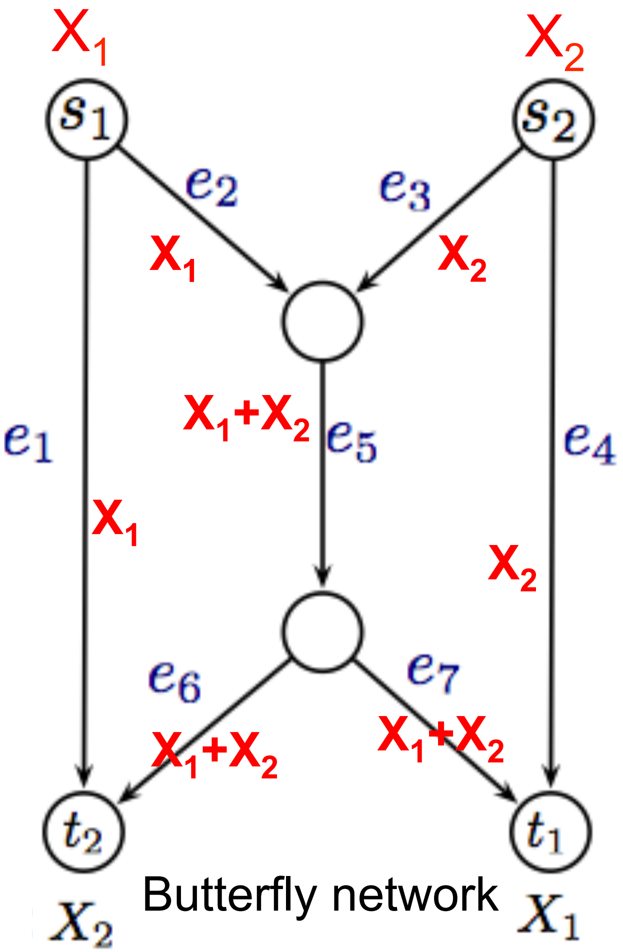
Equivalence bw Index Coding & Network Coding



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 a index code of length L, and vice versa. (same block length, same error probability).

Example



Butterfly network

$Y_{e1}+X_1$
$Y_{e2}+X_1$
$Y_{e3}+X_2$
$Y_{e4}+X_2$
$Y_{e5}+X_1+X_2$
$Y_{e6}+X_1+X_2$
$Y_{e7}+X_1+X_2$

server

X_1, X_2
$Y_{e1}, Y_{e2}, \ldots, Y_{e7}$
$H(Y_{ei})=c(e_i)=1$

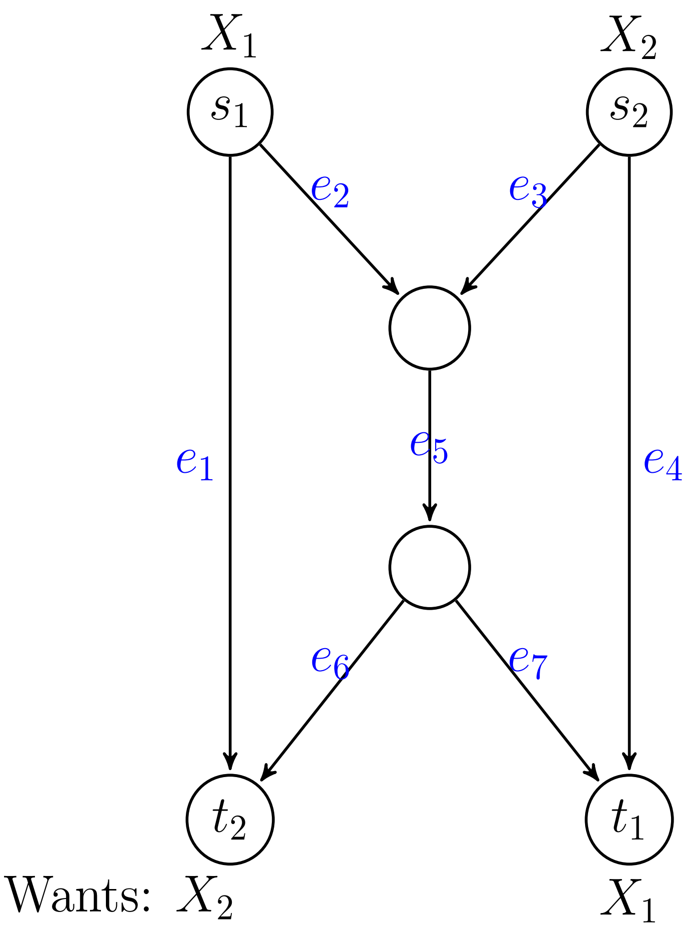
Terminal	Wants	Has
U_{e_1}	Y_{e_1}	X_1
U_{e_2}	Y_{e_2}	X_1
U_{e_3}	Y_{e_3}	X_2
U_{e_4}	Y_{e_4}	X_2
U_{e_5}	Y_{e_5}	$Y_{e_2}\ Y_{e_3}$

Terminal	Wants	Has
U_{e_6}	Y_{e_6}	Y_{e_5}
U_{e_7}	Y_{e_7}	Y_{e_5}

Equivalent index code

- All terminals in the index coding problem can decode
- Any linear network code gives an index code of length L=7

Butterfly Network



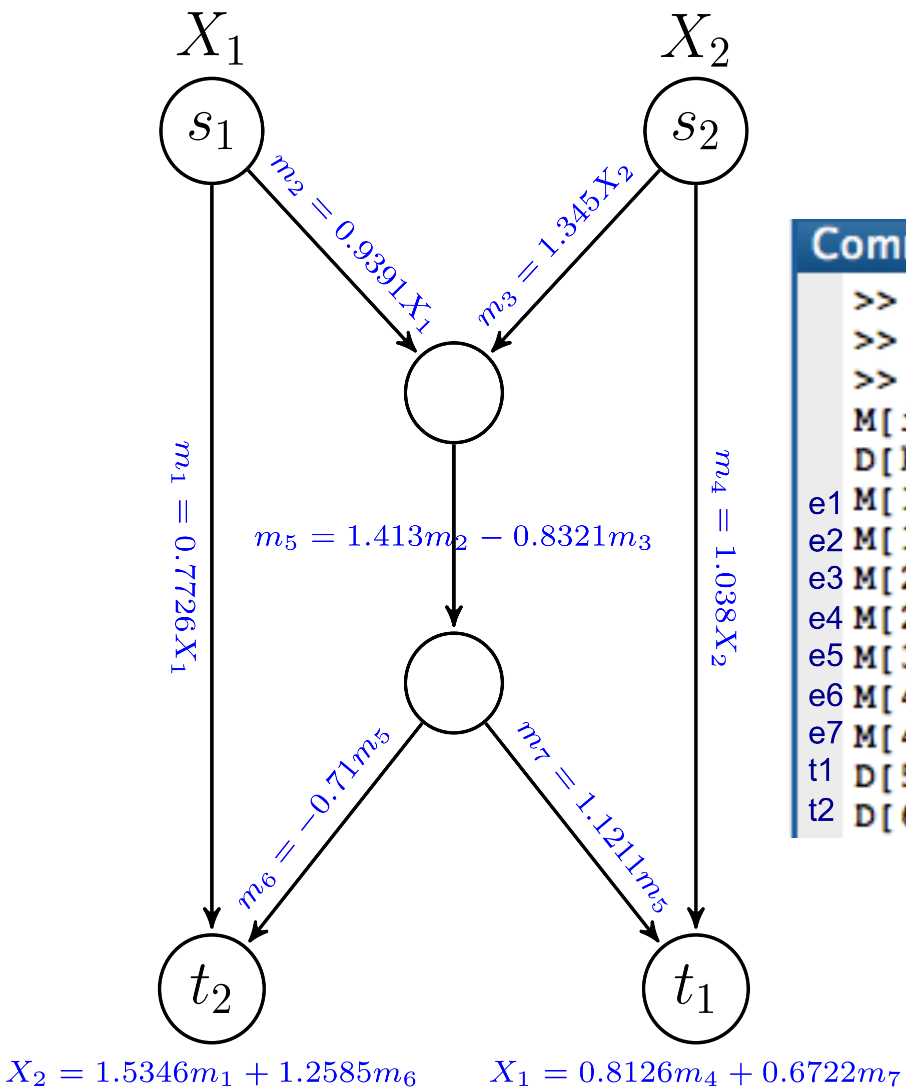
Command Window

```
>> 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)
```

Equivalent Index Coding Problem

```
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,3] = (-0.754565)*X1 ;
M[1,5] = (1.245211)*X1 ;
M[2,3] = (0.471860)*X2 ;
M[2,6] = (0.867619)*X2 ;
M[3,4] = (0.936511)*M[1,3] + (1.296273)*M[2,3] ;
M[4,5] = (1.812115)*M[3,4] ;
M[4,6] = (-0.976809)*M[3,4] ;
D[5] = (0.932065)*M[1,5] + (0.900983)*M[4,5] ;
D[6] = (0.990609)*M[2,6] + (1.438965)*M[4,6] ;
```

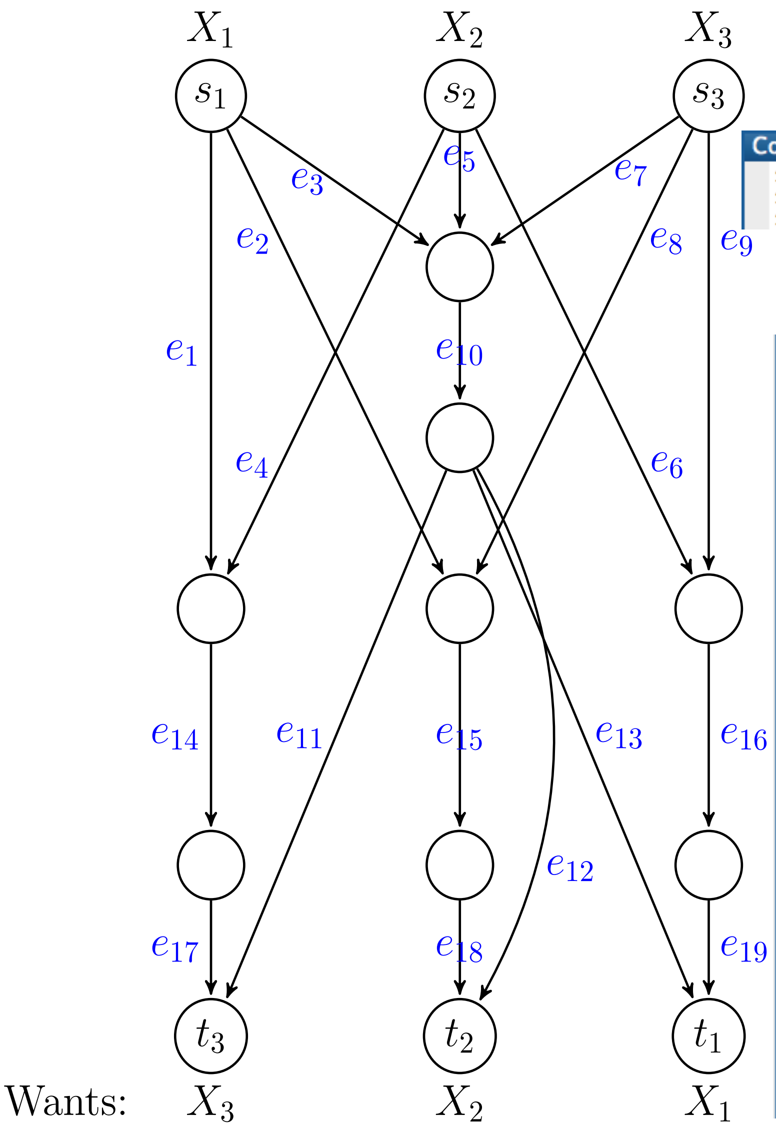
Index code



Command Window

```
   >> 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



Command Window

```
>> 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)
```

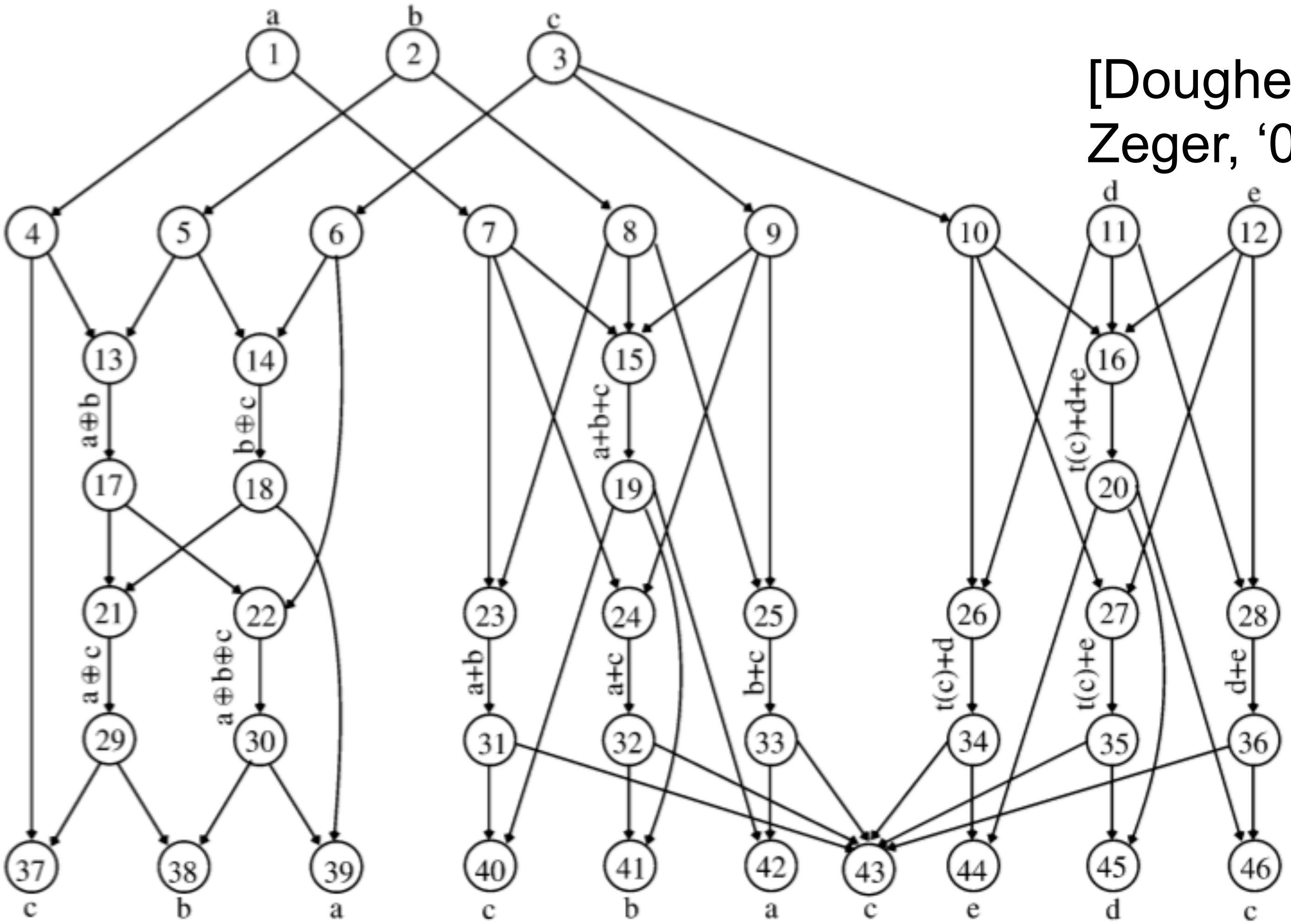
```
>> FindNetworkCode(Example2_Network,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,4] = (0.586472)*X1 ;
M[1,6] = (2.266747)*X1 ;
M[1,7] = (-0.063189)*X1 ;
M[2,4] = (0.589654)*X2 ;
M[2,6] = (2.666987)*X2 ;
M[2,8] = (0.911907)*X2 ;
M[3,4] = (1.119271)*X3 ;
M[3,7] = (0.655653)*X3 ;
M[3,8] = (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] ;
```

Solution of example 2

Command Window

```
>> 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] = (2.666987)*X2 ;
M[2,8] = (0.911907)*X2 ;
M[3,4] = (1.119271)*X3 ;
M[3,7] = (0.655653)*X3 ;
M[3,8] = (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

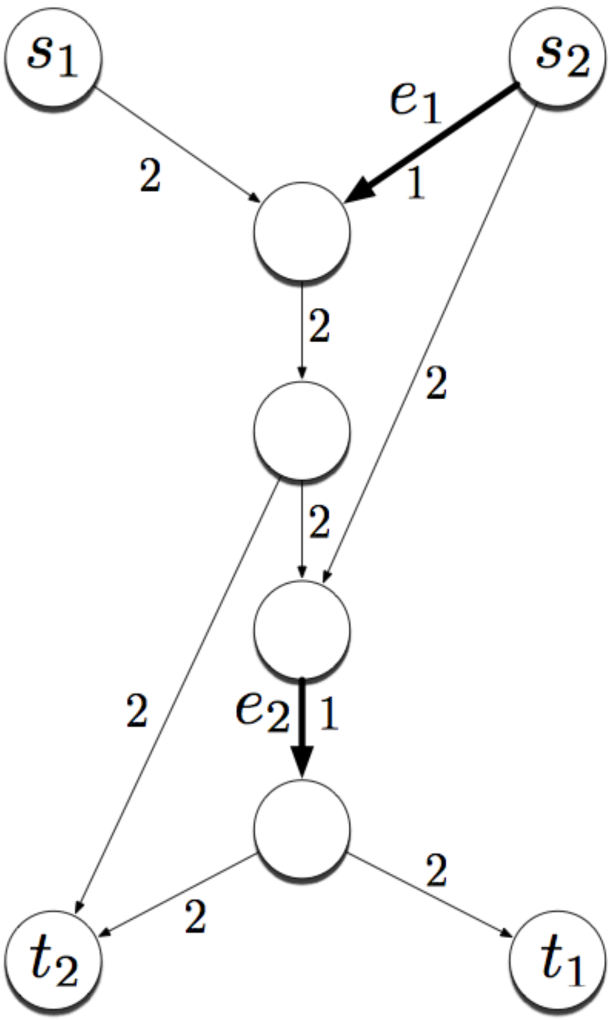


[Dougherty, Freiling, Zeger, '05]

Command Window

```
>> Network2=[1 4;2 4;2 5;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 3 2 1];
>> NC=FindNetworkCode(Network2,Demand)
Cannot find scalar linear network code.

NC =

     []
```

Example

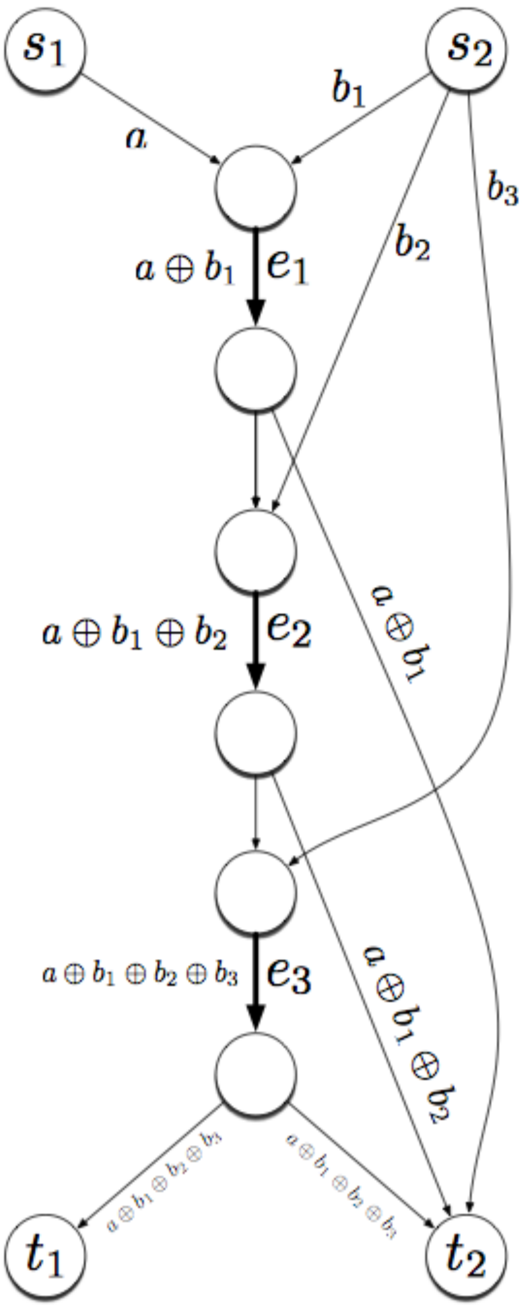


Command Window

```
>> Network_fig1=[1 3;2 3;2 5;3 4;4 7;4 5;5 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] ;
```

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

Examples

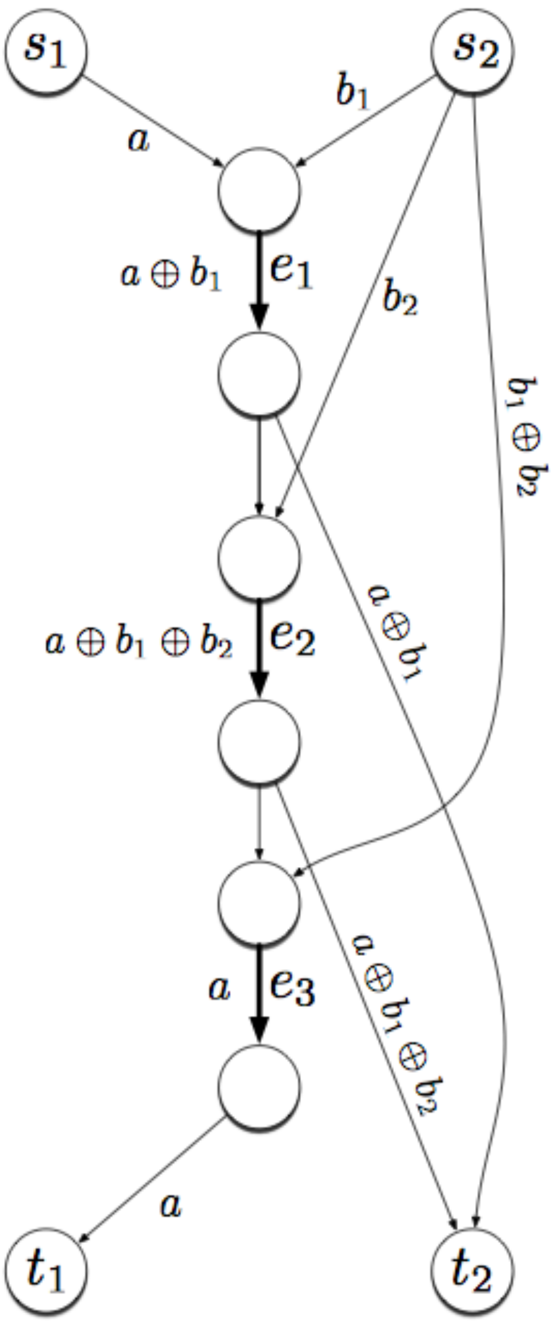


Command Window

```
>> Network_fig2b=[1 3;2 3;2 5;2 7;3 4;4 10;4 5;5 6;6 7;6 10;7 8;8 9];
>> Demand=[zeros(1,8) 1 2];
>> FindNetworkCode(Network_fig2b,Demand)
M[i,j] is the message on edge[i,j];
D[k] is the decoding message on node k;
M[1,3] = (1.839097)*X1 ;
M[2,3] = (1.246397)*X2 ;
M[2,5] = (1.196041)*X2 ;
M[2,7] = (0.958880)*X2 ;
M[3,4] = (-0.078045)*M[1,3] + (0.219144)*M[2,3] ;
M[4,5] = (0.432557)*M[3,4] ;
M[4,10] = (-0.266030)*M[3,4] ;
M[5,6] = (0.641976)*M[2,5] + (0.659312)*M[4,5] ;
M[6,7] = (-0.644002)*M[5,6] ;
M[6,10] = (1.009827)*M[5,6] ;
M[7,8] = (1.812203)*M[2,7] + (3.204077)*M[6,7] ;
M[8,9] = (3.814314)*M[7,8] ;
D[9] = (3.200609)*M[8,9] ;
D[10] = (1.249332)*M[4,10] + (1.279767)*M[6,10] ;
```

S. Kamath, Tse, Anantharam, "Generalized Network Sharing Outer Bound and the Two-Unicast Problem", 2011.

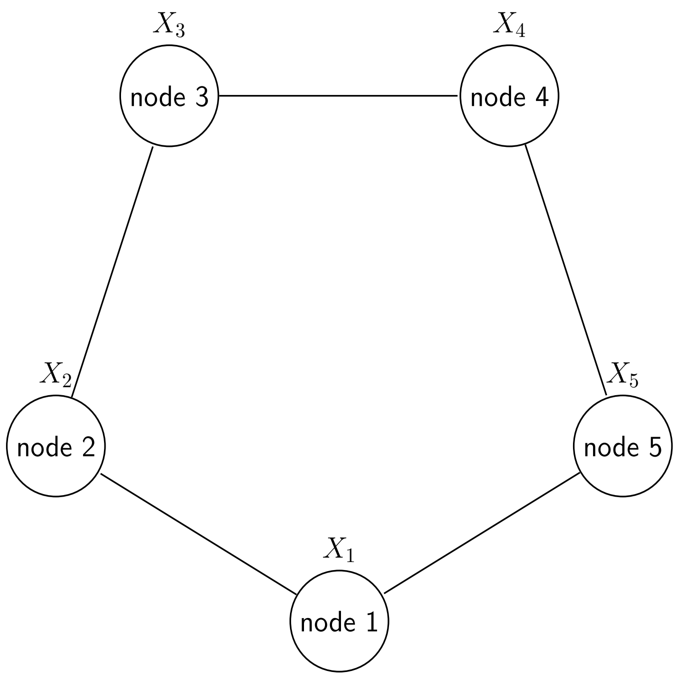
Examples



Command Window

```
>> 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] ;
```

Locally Repairable Code

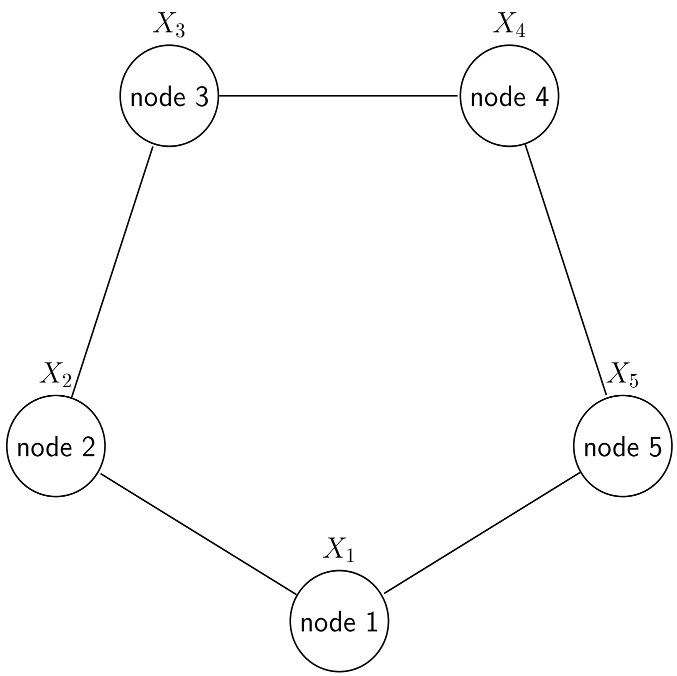


- Constructing Linear Repairable Codes* is equivalent to constructing linear index codes
- [Mazumdar '14],[Shanmugam, Dimakis'14]

Command Window

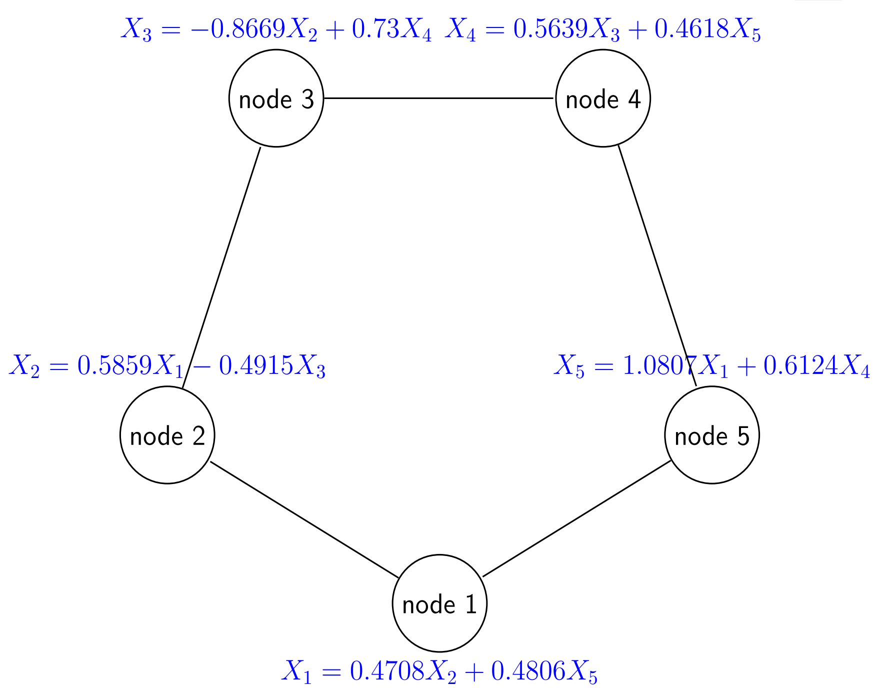
```
>> Pentagon=[1 2;2 3;3 4;4 5;1 5];
>> [StorageCode,FileSize]=FindLRC(Pentagon)

StorageCode =

         0    0.4708         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 =
```

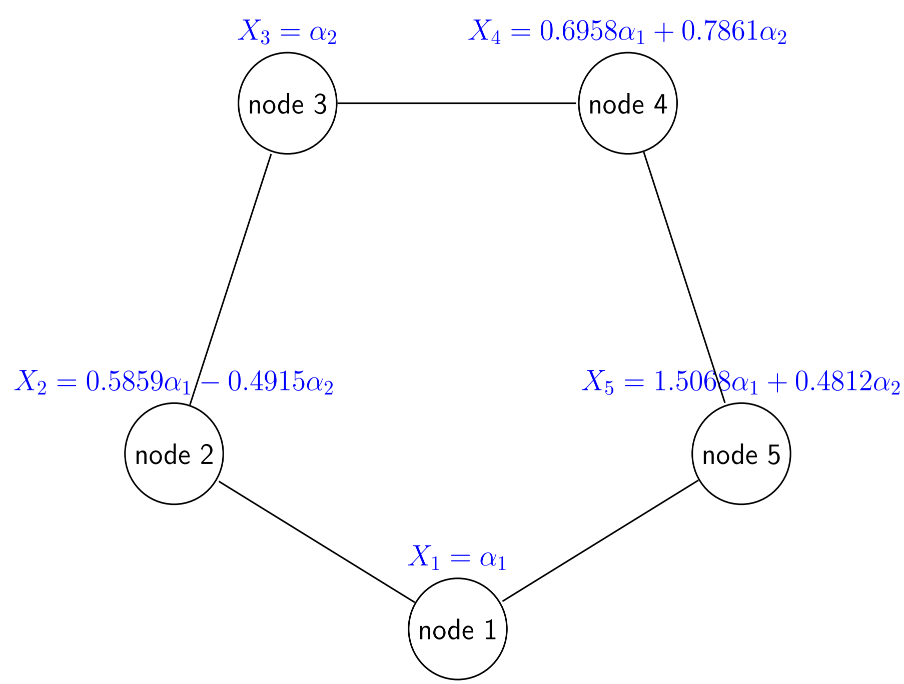
Locally Repairable Code



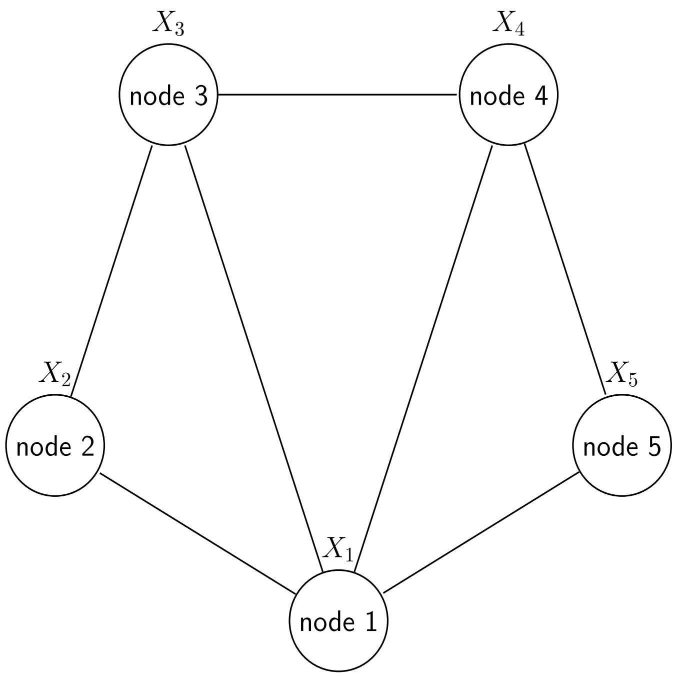
Command Window

```
>> 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
```



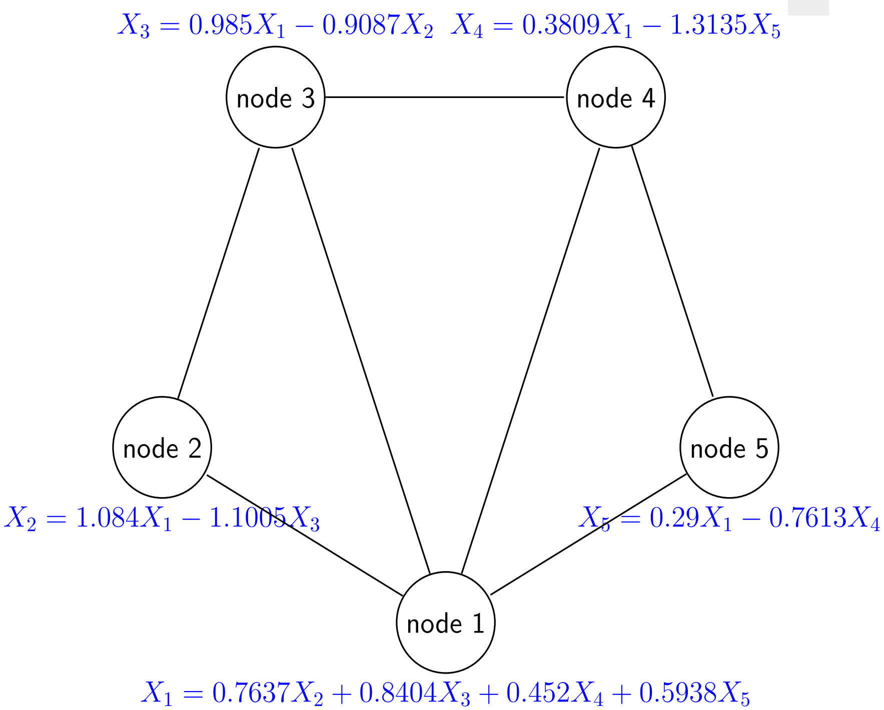


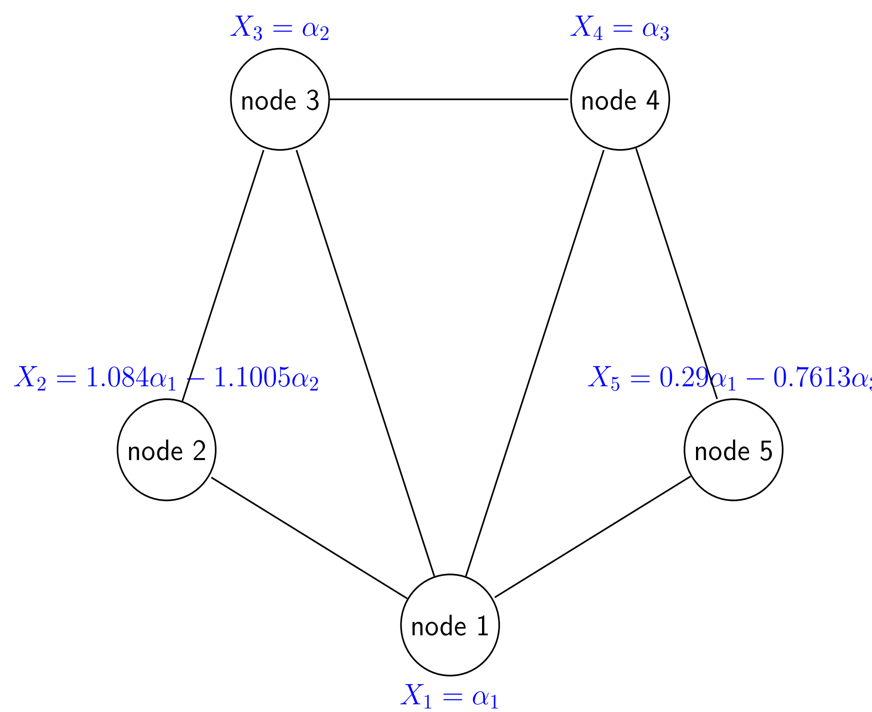
Another Example



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     0.0000          0     1.3135
   0.2900          0          0     0.7613          0


FileSize =

     3
```



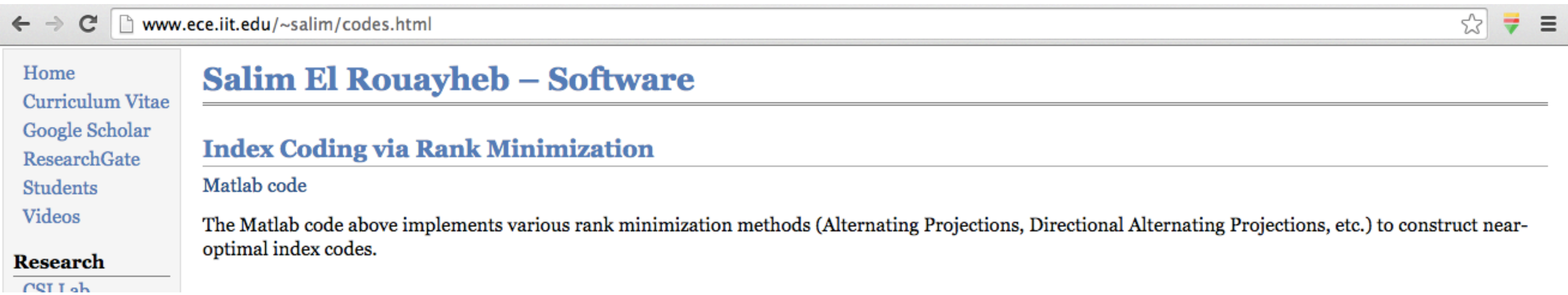


Concluding Remarks

- 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?

Code Available Online

www.ece.iit.edu/~salim/codes.html



Full Paper available on Arxiv.

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,

$$\|\mathbf{X} - \hat{\mathbf{X}}\| \leq \epsilon X_{\max}\sqrt{n}. \tag{5}$$

$$\begin{aligned}
\|\mathbf{X} - \hat{\mathbf{X}}\| &= \|\mathbf{X} - M^* A^\dagger A\mathbf{X} - M^* \circ \Phi\mathbf{X}\| && (10)\\
&= \|\mathbf{X} - M^*\mathbf{X} - M^* \circ \Phi\mathbf{X}\| && (11)\\
&= \|(I + M^* \circ \Phi - M^*)\mathbf{X}\| && (12)\\
&= \|(M_{\mathcal{D}} - M^*)\mathbf{X}\| && (13)\\
&\leq \|M_{\mathcal{D}} - M^*\|\|\mathbf{X}\| && (14)\\
&= \|U \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_{n-r^*} \end{bmatrix} V^T\| X_{\max}\sqrt{n} && (15)\\
&\leq \sigma_{r^*+1} X_{\max}\sqrt{n} && (16)\\
&\leq \epsilon X_{\max}\sqrt{n}. && (17)
\end{aligned}$$

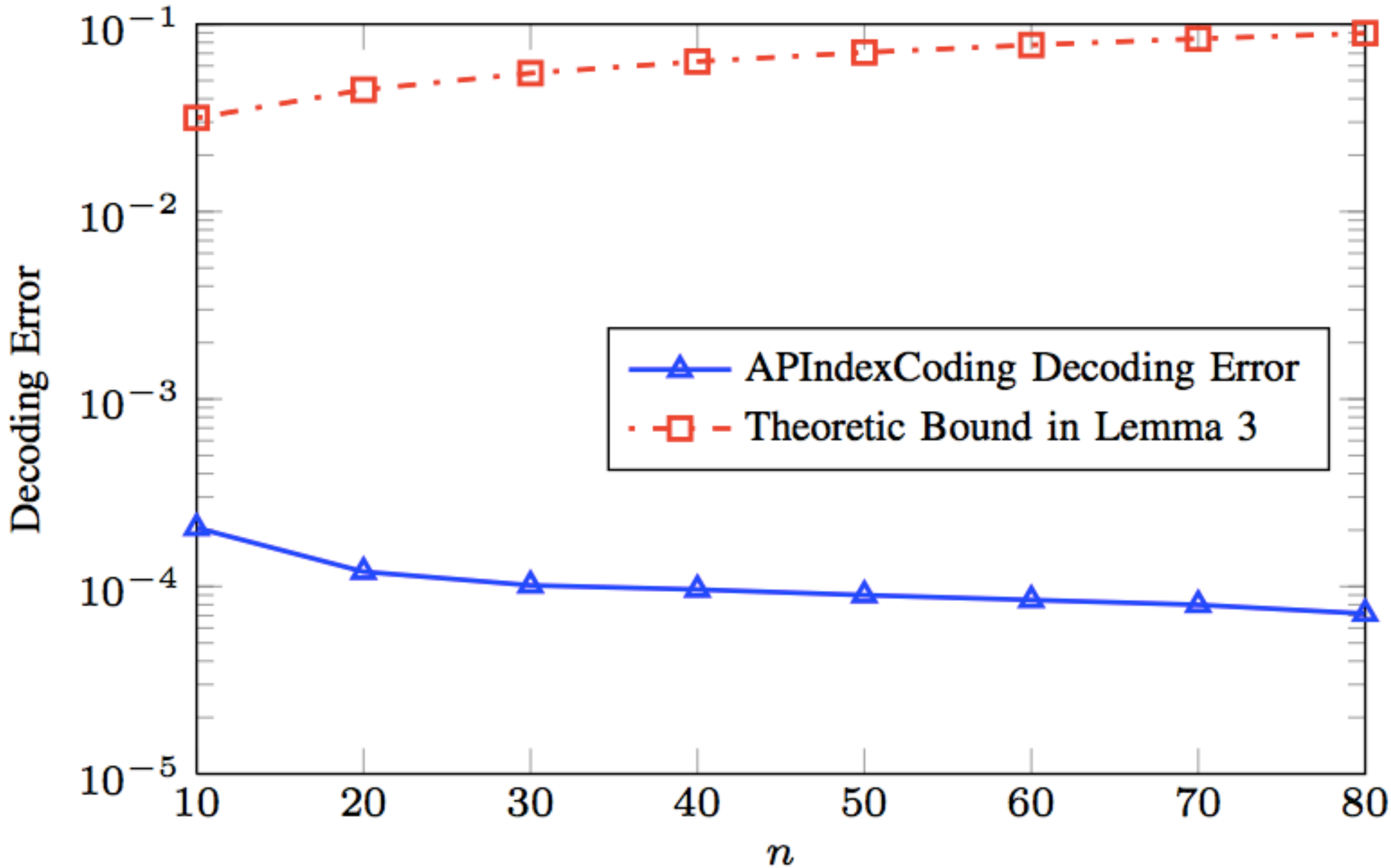


Fig. 12: Average decoding error $\|\mathbf{X} - \hat{\mathbf{X}}\|$ in APIndexcoding on random undirected graphs when $p = 0.2$, $\epsilon = 0.001$ and $X_i \in [-10, 10]$ ($X_{\max} = 10$).