Examples on using APIndexCoding and FindNetworkCode MATLAB functions

Xiao Huang and Salim El Rouayheb

1 APIndexCoding function

In this section, we illustrate through an example (Fig. 1) how to use the MATLAB function APIndexCoding to find a scalar linear solution for a given index coding problem.

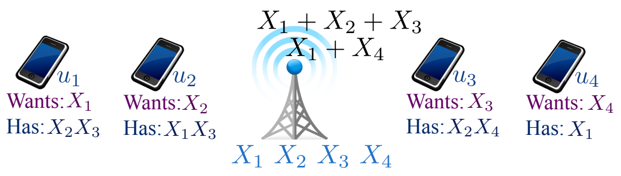


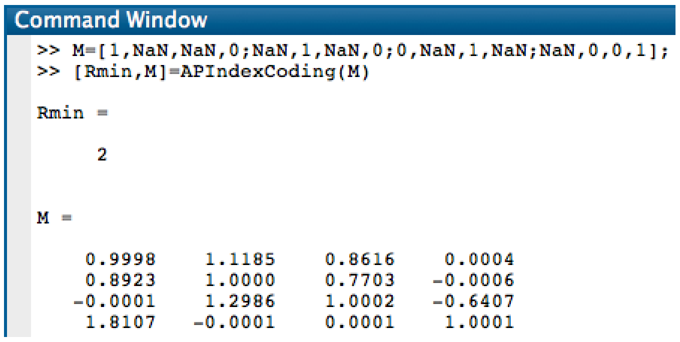
Figure 1: An index coding example.

The index coding problem of Fig. 1 can be seen as a rank minimization problem of the following matrix

$$M = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \end{array} \\ \begin{pmatrix} 1 & * & * & 0 \\ * & 1 & * & 0 \\ 0 & * & 1 & * \\ * & 0 & 0 & 1 \end{pmatrix} \end{array} .$$

So we could use M as input and get the optimal scalar linear solution. The MATLAB inputs are shown as follows, and the output is depicted in Fig. 2.

```
M = [1 NaN NaN 0; NaN 1 NaN 0; 0 NaN 1 NaN; NaN 0 0 1];
[Rmin, M] = APIndexCoding(M)
```



```
Command Window
>> M=[1,NaN,NaN,0;NaN,1,NaN,0;0,NaN,1,NaN;NaN,0,0,1];
>> [Rmin,M]=APIndexCoding(M)

Rmin =

     2


M =

    0.9998    1.1185    0.8616    0.0004
    0.8923    1.0000    0.7703   -0.0006
   -0.0001    1.2986    1.0002   -0.6407
    1.8107   -0.0001    0.0001    1.0001
```

Figure 2: Screenshot from MATLAB showing the optimal scalar linear solution of the index coding problem in Fig. 1 corresponding to a completed matrix M of rank 2.

This is the default way of using the function APIndexCoding. Check the help section in the MATLAB code for more details on the different options that this function offers.

2 FindNetworkCode Function

In this section, we show by constructing a scalar linear index code for the well known “Butterfly” network (Fig. 3) how to use the FindNetworkCode function. For more details about the function, please check the help section of FindNetworkCode in MATLAB.

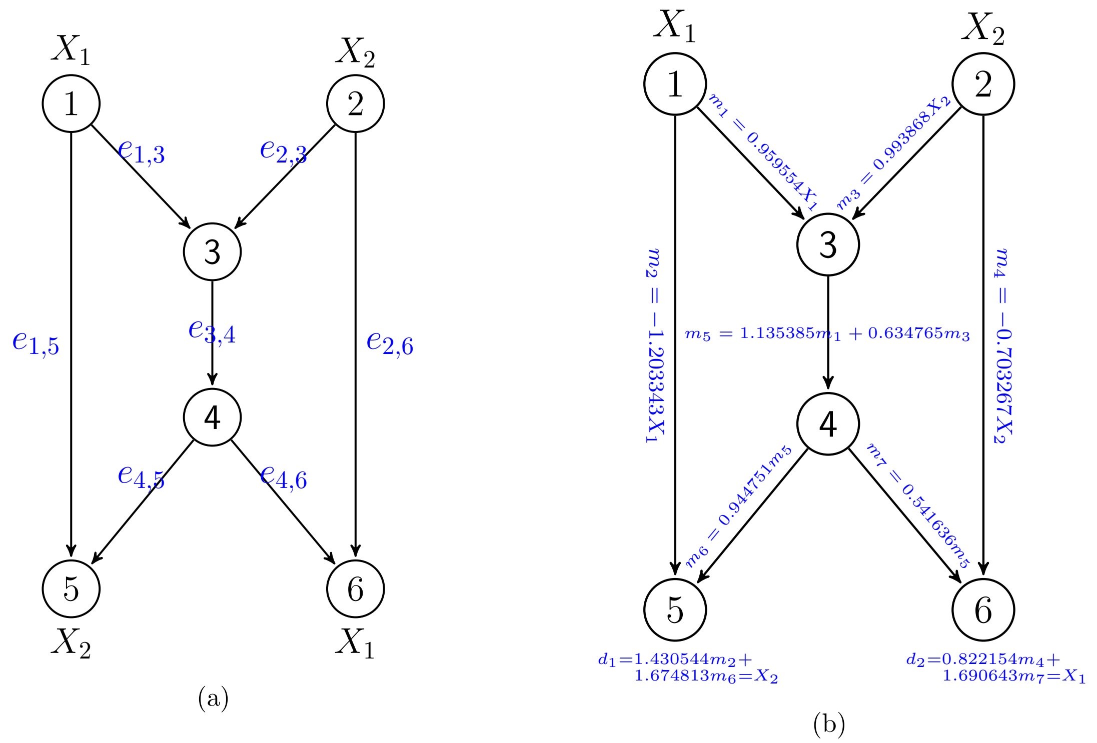
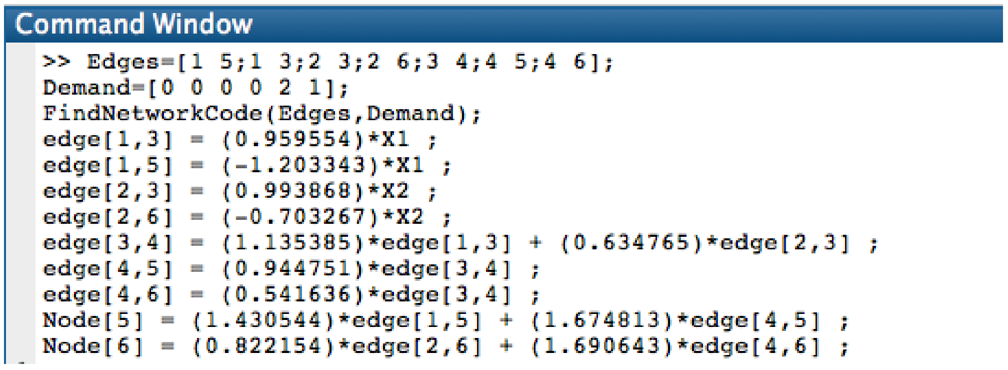


Figure 3: (a) The Butterfly network and (b) its corresponding network code obtained from function FindNetworkCode.

Consider the Butterfly network depicted in Fig. 3 (a). To generate a scalar linear index code for this network, define “Edges” as a 7×2 matrix, 7 being the total number of edges. Where in each row the edge e_{ij} is represented by i in the first column and j in the second column. Create a “Demand” vector where the value in the i^{th} entry is the number of the source that terminal i is interested in receiving its packet and is 0 if node i is not a terminal. Run the function as shown in the following code, the output is depicted in Fig. 5.

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

Updates: The screenshots here are from the old FindNetworkCode function. The new function will first try to find a solution to the equivalent index coding problem over $GF(2)$ using the LDG method from Birk and Kol. If it does not succeed it will try APIndexCoding to find a code over



```
>> Edges=[1 5;1 3;2 3;2 6;3 4;4 5;4 6];
Demand=[0 0 0 0 2 1];
FindNetworkCode(Edges,Demand);
edge[1,3] = (0.959554)*X1 ;
edge[1,5] = (-1.203343)*X1 ;
edge[2,3] = (0.993868)*X2 ;
edge[2,6] = (-0.703267)*X2 ;
edge[3,4] = (1.135385)*edge[1,3] + (0.634765)*edge[2,3] ;
edge[4,5] = (0.944751)*edge[3,4] ;
edge[4,6] = (0.541636)*edge[3,4] ;
Node[5] = (1.430544)*edge[1,5] + (1.674813)*edge[4,5] ;
Node[6] = (0.822154)*edge[2,6] + (1.690643)*edge[4,6] ;
```

Figure 4: Screenshot from MATLAB showing the resulting index code.

the reals. The way to call the function is still the same.

2.1 Output interpretations

In this section, we interpret the given results. First, notice that for some networks scalar linear codes are not optimal. In such case, the function FindNetworkCode will output "Cannot find scalar linear network code".

When scalar linear codes are optimal (e.g. butterfly network in Fig 3), we may get the results in Fig. 5. Now each "edge$[i, j]$" can be seen as the encoding of the message m_k sent through edge $e_{i,j}$ as a function of the incoming messages and the data packets. For instance, the output in Fig. 5 can be written as follows:

$$\begin{aligned}
m_1 &= e_{1,3} = 0.959554X_1, \\
m_2 &= e_{1,5} = -1.203343X_1, \\
m_3 &= e_{2,3} = 0.993868X_2, \\
m_4 &= e_{2,6} = -0.703267X_2, \\
m_5 &= e_{3,4} = 1.135385m_1 + 0.634765m_3, \\
m_6 &= e_{4,5} = 0.944751m_5, \\
m_7 &= e_{4,6} = 0.541636m_5.
\end{aligned}$$

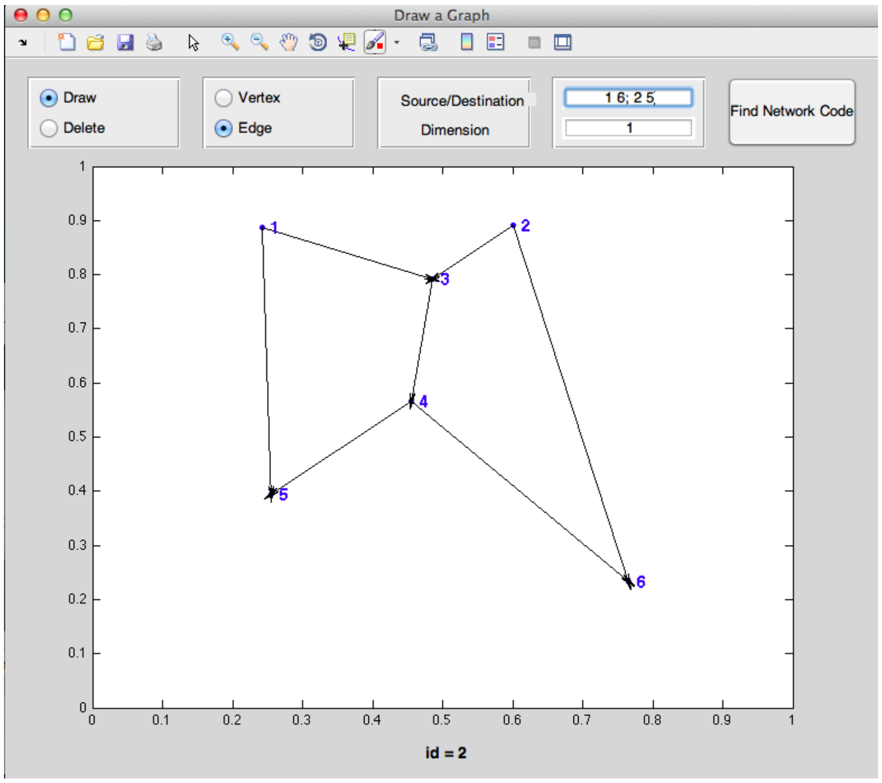
However, each "Node$[i]$" can be seen as the decoding function of the terminal t_l. Hence,

$$\begin{aligned}
d_1 &= \text{Node}[5] = 1.430544m_2 + 1.674813m_6 = X_2, \\
d_2 &= \text{Node}[6] = 0.822154m_4 + 1.690643m_7 = X_1.
\end{aligned}$$

And the resulting network code is depicted in Fig. 3 (b).

3 Graphical User Interface

The Matlab file is graph_gui.m. The GUI in the figure below allows the user to draw the network and call the FindNetworkFunction by hitting the FindNetworkCode Button. The demands are specified in the corresponding text box under the following format: “source1 destination1; source2 destination2; ...” The dimension text box takes as an input the desired dimension of the vector linear solution. However, in the current version this number is ignored and always taken to be 1 (i.e. the function tries to construct a scalar linear network code).


Draw a Graph

[✓ Draw] Delete

Vertex [✓ Edge]

Source/Destination: 1 6; 2 5

Dimension: 1

Figure 5: GUI for the FindNetworkCode function