An Extended Least Difference Greedy Clique-Cover Algorithm for Index Coding

Sangwoon Kwak, Jungho So and Youngchul Sung[†]
Dept. of Electrical Engineering
KAIST
Daejeon, Korea, 305-701
Email: {sw.kwak@, jhso@ and ysung@ee.}kaist.ac.kr

***Abstract*—In this paper, linear binary index coding is considered. It is shown that the minimum clique-cover heuristic algorithm can provide an efficient way to solving linear binary index coding problems. Based on the least difference greedy (LDG) clique-cover algorithm, an existing minimum clique-cover algorithm for index coding, proposed by Birk and Kol [1], [2], we develop an extended LDG algorithm by considering a transpose index coding model and cycle detection in the side information graph. Numerical results show that the proposed algorithm considerably outperforms the conventional LDG algorithm in terms of the number of transmissions.**

I. Introduction

Motivated by Birk and Kol [1], [2], coding-on-demand by an informed-source over a broadcast channel, also known as index coding, has attracted much attention from the research community in recent years [3]. The index coding problem [4] that we consider in this paper is described as follows:

Definition 1 (The considered index coding problem): A server communicates with n receivers $R = \{r_1, r_2, \cdots, r_n\}$ through an errorless broadcast channel, and the server holds a set of n binary variables $X = \{x_1, x_2, \cdots, x_n\} \in \{0,1\}^n$. Each receiver r_i requires x_i and has prior side information about the data X denoted by $X[N(i)]$, where $N(i)$ is the index set of side information at r_i. The goal of index coding is to minimize the number of transmissions for given side information index sets $N(1), N(2), \cdots, N(n)$.

Although the index coding problem was first introduced for satellite networks, the index coding technique can be applied to numerous applications including distributed storage systems, which have become important in recent years with the advent of social media. In the repair process of distributed storage systems with local hierarchy [5], [6], it is a challenging and important problem to reduce the number of global transmission between different local areas. Assuming the local transmission is cost-free, the repair problem of multiple failures can be regarded as an index coding problem. Thus, developing an efficient index coding algorithm is important to practical distributed storage systems in real world. In this paper, we investigate the LDG algorithm [1], [2], an efficient linear index coding algorithm, and propose an extended LDG algorithm by developing two extensions: *column merging heuristic* and *cycle detection*.

† This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2013R1A1A2A10060852).

A. Related work

In [4], it is shown that an index coding problem can be represented by a unique directed graph called a side information graph $G(V,E)$. Using the side information graph, Bar-Yoseef *et al.* completely characterized the optimal length of a linear index code for a given index coding problem [4]. For a given index coding problem represented by a directed graph G, there is a minimum code length for linear index coding that can be expressed by a graph functional. Although the authors in [7] showed that nonlinear index coding can outperform linear index coding for some problems, there are many cases in which still linear index coding is optimal [4]. There exist several algorithms for linear index code design, and some of these algorithms will be explained briefly in Section II-A.

II. Background

An index coding problem described in Definition 1 can be represented by a directed graph $G(V,E)$ as follows [4]:

Definition 2 (Side Information Graph): [4] The *side information graph* $G(V,E)$ with a vertex set V and an edge set E is defined as follows:

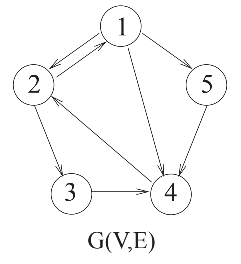
1) Construct n vertices. The i-th vertex (node) represents the i-th receiver r_i and the corresponding requested block x_i simultaneously.
2) Construct a directed edge (u,v) if and only if r_i knows x_j, i.e., $j \in N(i)$.

Bar-Yossef *et al.* obtained the minimum length for a linear index code represented by a directed graph $G(V,E)$ based on the concept of fitting matrices for given $G(V,E)$ defined as follows [4]:

Definition 3 (Fitting Matrices): For a given directed graph G we say that a binary $n \times n$ matrix $\mathbf{F} = (f_{ij})$ *fits* G if the following conditions hold for all i and j:

1) $f_{ii} = 1$,
2) $f_{ij} = 0$ whenever (i,j) is not an edge of G.

Then, the minimum length for a linear index code for $G(V,E)$ is given by a graph functional $minrk_2(G) \triangleq$



$$\mathbf{F}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{F}_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Fig. 1. The side information graph and fitting matrices for the Example 1

$\min\{rank_2(\mathbf{F}) \mid \mathbf{F} \text{ fits } G\}$ [4], and a matrix fitting G whose rank is $minrk_2(G)$ provides an optimal linear code for the index coding problem represented by the directed graph G. (For a fitting matrix $\mathbf{F}$ for G, $E(X) = [y_1, \cdots, y_{rank(\mathbf{F})}]$ given by

$$\begin{bmatrix} y_1 \\ \vdots \\ y_{rank(\mathbf{F})} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{rank(\mathbf{F})} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{1}$$

provides a linear index code for G, where $\mathbf{b}_1, \cdots, \mathbf{b}_{rank(\mathbf{F})}$ are a set of row vectors that spans the row space of $\mathbf{F}$ [4].)

To illustrate this, consider the following example.

Example 1: A server wants to broadcast an input $X \in \{0,1\}^5$ and five receivers $r_1, r_2, \cdots$, and r_5 request $x_1, x_2, \cdots$, and x_5, respectively. The side information index sets are given by

$$\begin{aligned} N(1) = \{2,3,4\}, N(2) = \{1,3\}, N(3) = \{4\}, \\ N(4) = \{2\}, N(5) = \{4\} \end{aligned} \tag{2}$$

Then, the side information graph G is constructed as shown in Fig. 1, and $\mathbf{F}_1$ and $\mathbf{F}_2$ are two examples of fitting matrices. Using an undetermined element $*$, all fitting matrices for a given side information graph G can be represented by a unique square matrix $\mathbf{A} \in \{0,1,*\}^{n \times n}$. For the considered example, the matrix representation of G is described by

$$\mathbf{A} = \begin{bmatrix} 1 & * & 0 & * & * \\ * & 1 & * & 0 & 0 \\ 0 & 0 & 1 & * & 0 \\ 0 & * & 0 & 1 & 0 \\ 0 & 0 & 0 & * & 1 \end{bmatrix}. \tag{3}$$

Since a matrix fitting G whose rank is $minrk_2(G)$ provides an optimal linear code for the index coding problem represented by the directed graph G, the linear binary index code design problem can be reduced to a *low rank matrix completion (LRMC) problem* over $GF(2)$. However, it is known that LRMC is NP-hard [8]. Thus, when the problem size is large, solving the problem based on LRMC is difficult. Thus, several researchers proposed heuristic algorithms for linear index coding and some of the well known heuristic algorithms are provided in the below.

A. Existing Algorithms for Index Coding

Most of the existing algorithms are based on minimum clique-cover heuristic to devise an efficient method to minimize the number of transmissions for a given linear index coding problem, since a clique can be covered by a single transmission.

1) The least difference greedy (LDG) clique-cover algorithm: For a given side information graph, there exist many different ways to clique-cover the graph. Consider the directed graph in Fig. 2 and consider two different ways of clique-cover as shown in methods 1 and 2 in Fig. 2. One can easily see that Method 1 of clique-cover yields a shorter code length. Birk and Kol [1], [2] proposed a heuristic way to find a better way to clique-cover by defining some distance between two rows of a fitting matrix. From a matrix completion perspective, clique-covering can be regarded as merging of the corresponding rows. The distance between two rows in a fitting matrix is defined as the sum of inter-entry distance, where the inter-entry distance d is defined as [2]

$$\begin{gathered} d(0,0) = d(1,1) = d(*,*) = 0, \\ d(0,*) = d(1,*) = 1, \text{ and } d(0,1) = \infty. \end{gathered} \tag{4}$$

Based on the defined inter-row distance, the LDG algorithm finds the minimum (finite) distance among all possible pairs of two rows in a fitting matrix and merge two rows with the minimum inter-row distance, and then iterates this procedure until all inter-row distances become infinite. At the step of merging two rows, $*$ in the fitting matrix is determined as either 0 or 1, and the two nodes corresponding to the two merged rows become a clique. In this way, one can cover the whole graph with a smaller number of cliques.

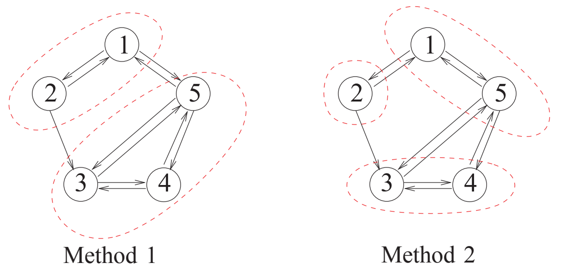


Fig. 2. Two different clique-cover methods for a given directed graph

2) Maximum matching: The maximum matching is a clique-cover method to maximize the number of cliques of size 2. Since a clique of size 2 can be covered by a single transmission, the maximum matching can be a heuristic way to solve linear index coding problems. We consider the existing maximum matching algorithm proposed by Galil [9] to performance comparison.

3) Color saving heuristic: The color saving heuristic algorithm [3] first finds cliques of size three and then computes a maximum matching of the resulting graph. In Section IV, we implement the first step by brute force and compute the maximum matching by the algorithm in [9].

III. The Proposed Index Coding Algorithm: An Extended LDG Algorithm

In this section, we develop two new extensions on the existing LDG algorithm in [2]. In the first subsection, we propose an extension named column-merging heuristic by considering the LDG process on the transpose matrix. In the second, we present an observation that there is still an opportunity to additionally reduce the rank of the matrix obtained by the LDG algorithm, if there is a cycle in the resulting side information graph.

A. Column-merging heuristic

Note that a linear binary index coding problem following Definition 1 can fully be represented by a square matrix $\mathbf{A} \in \{0, 1, *\}^{n \times n}$ whose diagonal entries are all 1 and the other entries are 0 or *. Using this fact, we define the transpose problem of a given index coding problem as follows.

Definition 4 (Transpose Index Coding Problem): For a given linear binary index coding problem $\mathcal{I}$ that is represented by a square matrix $\mathbf{A}$, the *transpose index coding problem* $\mathcal{I}^T$ is defined as the problem represented by $\mathbf{A}^T$, where $\mathbf{A}^T$ is the transpose of $\mathbf{A}$.

It is easy to see that Definition 4 is valid since $\mathbf{A}^T$ also has all 1 diagonal entries and 0 or $*$ at the off-diagonal positions. We provide the relation between an index coding problem and its transpose index coding problem as introducing the following proposition.

Proposition 1: There exists a linear index code of length l for an index coding problem $\mathcal{I}$ if and only if there exists a linear index code of length l for $\mathcal{I}^T$.

Proof: Consider a linear index coding problem $\mathcal{I}$ that can be represented by a square matrix $\mathbf{A} \in \{0, 1, *\}^{n \times n}$. If there exists a linear index code of length l for $\mathcal{I}$, there exists a fitting matrix $\mathbf{A}_1$ for $\mathcal{I}$ such that $rank(\mathbf{A}_1) = l$. Then, we can obtain a fitting matrix $\mathbf{A}_1^T$ for $\mathcal{I}^T$ such that $rank(\mathbf{A}_1^T) = l$, which represents a linear index code of length l for $\mathcal{I}^T$. The converse is trivial from the fact that $(\mathcal{I}^T)^T = \mathcal{I}$. ■

From Proposition 1, the row-combing LDG algorithm is applied to $\mathbf{A}^T$ first and the resulting matrix is transposed to obtain a fitting matrix for the original problem. In some cases, this approach yields a better node-merging result. In fact, at each merging step of the LDG algorithm, this alternative approach can be considered. To illustrate this, consider the following example:

Example 2: Consider a linear binary index coding problem $\mathcal{I}$ described by a 4×4 matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & * & 0 \\ * & 1 & 0 & * \\ * & * & 1 & * \\ * & * & * & 1 \end{bmatrix}. \tag{5}$$

Applying the LDG algorithm to $\mathbf{A}$,

	r_1	r_2	r_3	r_4
r_1	-	∞	4	∞
r_2	-	-	∞	3
r_3	-	-	-	$\underline{2}$
r_4	-	-	-	-

$\longrightarrow$

	r_1	r_2	r_3, r_4
r_1	-	∞	∞
r_2	-	-	∞
r_3, r_4	-	-	-

(6)

we obtain the resulting matrix $\mathbf{A}_{\text{LDG}(\mathcal{I})}$:

$$\mathbf{A}_{\text{LDG}(\mathcal{I})} = \begin{bmatrix} 1 & 0 & * & 0 \\ * & 1 & 0 & * \\ * & * & 1 & 1 \\ * & * & 1 & 1 \end{bmatrix}. \tag{7}$$

As we can see in (7) and the corresponding row distance table (6), in this case, only one row-merging occurs and this only guarantees the reduction of the rank by one. However, if we apply the LDG algorithm to $\mathcal{I}^T$, there still exist a finite number in the row distance table after the first row-merging as seen in (8), and thus we can have one more row-merging. This guarantees the reduction of the matrix rank by two.

	r_1	r_2	r_3	r_4
r_1	-	∞	3	∞
r_2	-	-	∞	$\underline{2}$
r_3	-	-	-	4
r_4	-	-	-	-

$\longrightarrow$

	r_1	r_2, r_4	r_3
r_1	-	∞	$\underline{4}$
r_2, r_4	-	-	∞
r_3	-	-	-

(8)

Finally, we obtain $\mathbf{A}_{\text{LDG}(\mathcal{I}^T)}$.

$$\mathbf{A}_{\text{LDG}(\mathcal{I}^T)} = \begin{bmatrix} 1 & 0 & 1 & * \\ 0 & 1 & * & 1 \\ 1 & 0 & 1 & * \\ 0 & 1 & * & 1 \end{bmatrix}. \tag{9}$$

Note that transposing the matrix $\mathbf{A}$, applying one row-merging step of the LDG algorithm, and transposing the resulting matrix back is equivalent to just column-merging in the original matrix $\mathbf{A}$. As shown in the example, this column merging can be more effective than row merging, depending on the number of $*$'s consumed for merging. Merging is simply making two rows (or two columns) the same by setting $*$ to 0 or 1 appropriately. Hence, if less $*$'s are consumed, we have more chance for a larger number of merging steps, i.e., larger rank reduction. Hence, at each merging step of the LDG algorithm, we consider both row-merging and column-merging and choose the merging requiring less $*$'s.

B. Cycle-of-three-nodes detection

We first present an example that shows the motivation of the second extension.

Example 3: Consider a linear binary index coding problem represented by a square matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & * & * & 0 \\ * & 1 & * & * \\ 0 & 0 & 1 & * \\ * & * & 0 & 1 \end{bmatrix}. \tag{10}$$

Before applying the LDG algorithm to $\mathbf{A}$, we present a solution matrix $\mathbf{A}_{\text{opt}}$ of which rank over $GF(2)$ is $minrk_2(G) = 2$,

$$\mathbf{A}_{\text{opt}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}. \tag{11}$$

Now, applying the LDG algorithm to $\mathbf{A}$, we have

$$\mathbf{A} = \begin{bmatrix} 1 & * & * & 0 \\ * & 1 & * & * \\ 0 & 0 & 1 & * \\ * & * & 0 & 1 \end{bmatrix} \longrightarrow \mathbf{A}_{\text{LDG}} = \begin{bmatrix} 1 & 1 & * & 0 \\ 1 & 1 & * & 0 \\ 0 & 0 & 1 & * \\ * & * & 0 & 1 \end{bmatrix}. \tag{12}$$

After the LDG algorithm yields $\mathbf{A}_{\text{LDG}}$ which has remaining $*$'s, the conventional LDG algorithm finally fills $*$'s in the merged rows with the same arbitrary value and fills other $*$'s arbitrarily. Since only one row-merging occurs in (12), the guaranteed rank reduction is one in this case. However, an important observation is that $\mathbf{A}_{\text{LDG}}$ still has a chance to become $\mathbf{A}_{\text{opt}}$ by filling out the remaining $*$'s in a smart way. Fig. 3 shows the merged directed graph for Example 3 after the merging steps of the LDG algorithm are finished. In the figure, it is seen that a cycle is formed after nodes 1 and 2 are merged into a clique by the LDG algorithm. Due to the cycle, there is an opportunity that the rows of $\mathbf{A}_{\text{LDG}}$ with the merged rows considered as one row can further be linearly dependent and the rank reduction at least by one can be attained. The

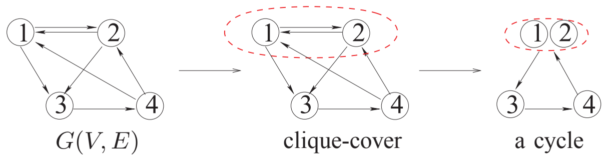


Fig. 3. The LDG algorithm process of Example 3.

purpose of the second extension is to find the possibility of linear dependence among the rows of an incomplete matrix $\mathbf{A}_{\text{LDG}}$ with the merged rows considered as one row and to fill out the remaining $*$'s so that the linear dependence is realized if it exists. Then, how can we find the possible linear dependence among the rows of $\mathbf{A}_{\text{LDG}}$ with the merged rows considered as one row just by observing $\mathbf{A}_{\text{LDG}}$?

To answer this question, consider $\mathbf{A}_{\text{opt}}$ and its first, third and fourth rows. (The first and second rows will be merged by the LDG algorithm.) We have

$$\begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}, \quad (13)$$

where $\oplus$ is the modulo-2 addition operator. That is, the first, third and fourth rows of $\mathbf{A}_{\text{opt}}$ are linearly dependent over $GF(2)$. Now consider the corresponding rows in $\mathbf{A}_{\text{LDG}}$ and the following equation corresponding to (13):

$$\begin{bmatrix} 1 & 1 & * & 0 \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 1 & * \end{bmatrix} = \begin{bmatrix} * & * & 0 & 1 \end{bmatrix}, \quad (14)$$

We need to check that (14) is valid with properly assigned $*$'s in (14). To do so, we define a *row addition operation* $\oplus^*$ as follows.

Definition 5 (Row Addition Operation with $*$): For a given square matrix $\mathbf{A} \in \{0,1,*\}^{n\times n}$, *the row addition operator* $\oplus^*$ *with* $*s$ is defined as follows.

$$\mathbf{A}_{i,k} \oplus^* \mathbf{A}_{j,k} = \begin{cases} \mathbf{A}_{i,k} \oplus \mathbf{A}_{j,k} & \mathbf{A}_{i,k} \neq * \text{ and } \mathbf{A}_{j,k} \neq * \\ * & \mathbf{A}_{i,k} = * \text{ or } \mathbf{A}_{j,k} = *, \end{cases} \quad (15)$$

where $\mathbf{A}_i$ and $\mathbf{A}_{i,k}$ denote the i-th row and the (i,k)-th entry of $\mathbf{A}$, respectively.

By the definition, the addition of $\mathbf{A}_1$ and $\mathbf{A}_3$ is now given by

$$\begin{bmatrix} 1 & 1 & * & 0 \end{bmatrix} \oplus^* \begin{bmatrix} 0 & 0 & 1 & * \end{bmatrix} = \begin{bmatrix} 1 & 1 & * & * \end{bmatrix}. \quad (16)$$

Since the added row has a finite inter-row distance (defined in (4)) with the $\mathbf{A}_4 = \begin{bmatrix} * & * & 0 & 1 \end{bmatrix}$, it is possible to make the added row the same $\mathbf{A}_4$. Both $\mathbf{A}_1 \oplus \mathbf{A}_3$ and $\mathbf{A}_4$ can be made as $[1,1,0,1]$. Thus, the linear dependence of any three rows $\mathbf{A}_i$, $\mathbf{A}_j$, and $\mathbf{A}_k$ of $\mathbf{A}_{\text{LDG}}$ can be checked by computing $\mathbf{A}_i \oplus^* \mathbf{A}_j$, computing the inter-row distance between $\mathbf{A}_i \oplus^* \mathbf{A}_j$ and $\mathbf{A}_k$, and checking the finiteness of the inter-row distance.

Proposition 2: Consider a linear index coding problem for a graph G that is represented by a square matrix $\mathbf{A} \in \{0,1,*\}^{n\times n}$. After applying the LDG algorithm to $\mathbf{A}$, we have $\mathbf{A}_{\text{LDG}}$, which can have remaining $*$'s. For a selected set of three rows of $\mathbf{A}_{\text{LDG}}$, if an added row of any two rows of the selected set has finite distance with the other row of the selected set, the set of three nodes which corresponds to the set of three rows forms a cycle in G.

Proof: We briefly sketch the proof due to space limitation. Since the merging steps of the LDG algorithm is finished, we have only two cases for a selected set of three rows, as shown in Fig. 4. Note that in Fig. 4 there cannot be a pair

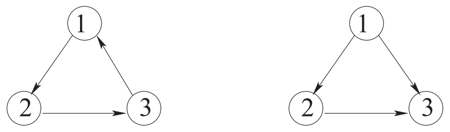


Fig. 4. Two non-isomorphic directed graphs with three nodes

of nodes with bidirectional links. If such a pair should exist, then the pair would form a clique and the merging steps of the LDG algorithm would reduce the pair to be one node. The linearly dependent case corresponds to the case of a cycle of three nodes in the left side in Fig. 4. (In the case of the graph in the right side of Fig. 4, the corresponding three rows cannot be made linearly dependent by any choice for $*$'s.) Thus, checking the linear dependence of three rows in $\mathbf{A}_{\text{LDG}}$ corresponds to detection of a cycle of three nodes. ■

If the probability that each pair of two nodes in the directed graph has an edge is $p < 1$, then the probability that an arbitrary set of three nodes forms a cycle is p^3, whereas in the case of a cycle with four nodes the probability is p^4. Thus, when p is sufficiently small, it is quite effective only to search cycles with three nodes. The proposed second extension to the original LDG algorithm is to search all cycles of three nodes existing in the directed graph of effective nodes in $\mathbf{A}_{\text{LDG}}$ after the merging steps of the LDG algorithm is finished. For the detected cycles with three nodes, we fill out the corresponding $*$'s to realize the linear dependence of the corresponding three rows. The proposed algorithm is given in Algorithm I.

IV. Numerical Results

To evaluate the performance of the proposed algorithm, we compare the algorithm with other existing index coding algorithms: the original LDG algorithm [2], the maximum matching [9], and the color saving heuristic [3]. We considered two network size of 50 nodes and 100 nodes. In each case, we randomly generated an edge for each pair of nodes in the network independently with probability P_{edge} 20 times. For each realized index coding network, we evaluated the performance of the considered linear index coding algorithms. The performance is measured by the length of the resulting index code. Figures 5 and 6 show the performance result averaged over the 100 random realizations for 50 nodes case and 40 random realizations for 100 nodes case. It is seen that the performance gain of the proposed algorithm over the existing algorithms in the reasonable range of $10^{-2} \le P_{edge} \le 10^{-1}$ is considerable. It is seen that roughly 10 % of the code length reduction is achieved. Note that the complexity increase of the proposed algorithm is not prohibitive. Search for cycles with three nodes requires complexity order of N^3 for network size N.

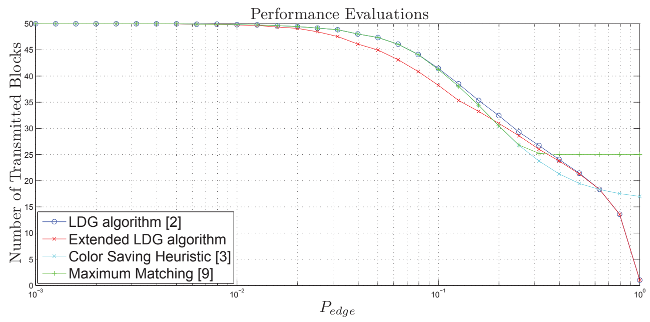


Fig. 5. Performance: Index coding network of 50 nodes

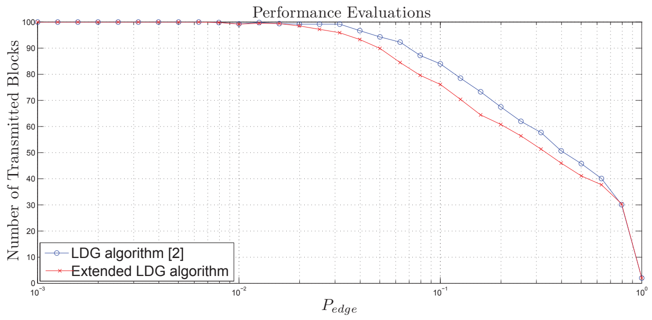


Fig. 6. Performance: Index coding network of 100 nodes

V. Conclusion

In this paper, we have considered the linear binary index coding problem. Motivated by practical needs for an efficient linear index coding algorithm, we have studied some existing index coding algorithms and have proposed an improved linear binary index coding algorithm by adding two extensions to the existing LDG algorithm: column-merging at each merging step and cycle-of-three-nodes detection at the end of the LDG algorithm. Numerical result shows that the gain of the proposed algorithm based on two simple heuristics is considerable.

References

[1] Y. Birk and T. Kol, "Informed-source coding-on-demand (iscod) over broadcast channels," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1257–1264 vol.3, 1998.

[2] Y. Birk and T. Kol, "Coding on demand by an informed source (iscod) for efficient broadcast of different supplemental data to caching clients," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2825–2830, 2006.

[3] M. A. R. Chaudhry and A. Sprintson, "Efficient algorithms for index coding," in *INFOCOM Workshops 2008, IEEE*, pp. 1–4, 2008.

[4] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *Information Theory, IEEE Transactions on*, vol. 57, no. 3, pp. 1479–1494, 2011.

[5] D. Papailiopoulos and A. Dimakis, "Locally repairable codes," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pp. 2771–2775, 2012.

[6] A. Rawat, O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[7] E. Lubetzky and U. Stav, "Nonlinear index coding outperforming the linear optimum," *Information Theory, IEEE Transactions on*, vol. 55, no. 8, pp. 3544–3551, 2009.

[8] B. Recht, M. Fazel, and P. Parrilo, "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM Review*, vol. 52, no. 3, pp. 471–501, 2010.

[9] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Comput. Surv.*, vol. 18, pp. 23–38, Mar. 1986.

Algorithm 1 The Proposed Extended LDG Algorithm

```
Require: A square matrix $\mathbf{A} \in \{0,1,*\}^{n\times n}$ which represents a linear binary index coding problem.
  repeat
    for $i = 1$ to $n-1$ do
      for $j = i+1$ to $n$ do
        $dr(i,j) = \sum_{k\in\text{columns}} d(\mathbf{A}_{i,k}, \mathbf{A}_{j,k})$
        $dc(i,j) = \sum_{k\in\text{rows}} d(\mathbf{A}_{k,i}, \mathbf{A}_{k,j})$
      end for
    end for
    $(i_1, j_1) = \arg\min_{(i,j):i<j,dr(i,j)\neq 0} dr(i,j)$
    $(i_2, j_2) = \arg\min_{(i,j):i<j,dc(i,j)\neq 0} dc(i,j)$
    if $dr(i_1,j_1) \le dc(i_2,j_2)$ and $dr(i_1,j_1) < \infty$ then
      for $k = 1$ to $n$ do
          $\mathbf{A}_{r,k} = \begin{cases} \mathbf{A}_{i_1,k} & \mathbf{A}_{i_1,k} = \mathbf{A}_{j_1,k} \\ \mathbf{A}_{i_1,k} & \mathbf{A}_{j_1,k} = * \\ \mathbf{A}_{j_1,k} & \text{otherwise.} \end{cases}$   (17)
      end for
      $\mathbf{A}_{i_1} \leftarrow \mathbf{A}_r$, $\mathbf{A}_{j_1} \leftarrow \mathbf{A}_r$
    else if $dc(i_2,j_2) < dr(i_1,j_1)$ then
      for $k = 1$ to $n$ do
          $\mathbf{A}_{k,c} = \begin{cases} \mathbf{A}_{k,i_2} & \mathbf{A}_{k,i_2} = \mathbf{A}_{k,j_2} \\ \mathbf{A}_{k,i_2} & \mathbf{A}_{k,j_2} = * \\ \mathbf{A}_{k,j_2} & \text{otherwise.} \end{cases}$   (18)
      end for
      $\mathbf{A}^T_{i_2} \leftarrow \mathbf{A}^T_c$, $\mathbf{A}^T_{j_2} \leftarrow \mathbf{A}^T_c$
    end if
  until $\min\left(dr(i_1,j_1), dc(i_2,j_2)\right) = \infty$
  for $cyc = 1$ to 2 do
    for $i = 1$ to $n-2$ do
      for $j = i+1$ to $n-1$ do
        for $k = 1$ to $n$ do
            $\mathbf{A}_{add(i,j),k} = \mathbf{A}_{i,k} \oplus^* \mathbf{A}_{j,k}.$   (19)
        end for
        repeat
          for $m = j+1$ to $n$ do
            $dr_m(m) = \sum_{k\in\text{columns}} d(\mathbf{A}_{add(i,j),k}, \mathbf{A}_{m,k})$
          end for
          $m_1 = \arg\min_{j<m\le n, dr_m(m)\neq 0} dr_m(m)$
          Make $\mathbf{A}_{add(i,j)}$ and $\mathbf{A}_{m_1}$ be same.
        until $dr_m(m)$ are 0 or $\infty$ for all $j < m \le n$
        if $cyc = 1$ then
          Fill out the $*$'s in $\mathbf{A}_i$ and $\mathbf{A}_j$ by Rule 1.
        else
          Fill out the $*$'s in $\mathbf{A}_i$ and $\mathbf{A}_j$ by Rule 2.
        end if
      end for
    end for
  end for
  (Filling rules)
      (Rule 1) If any two elements of $\{\mathbf{A}_{i,k}, \mathbf{A}_{j,k}, \mathbf{A}_{add(i,j),k}\}$ are already determined with one element undetermined, determine the remaining $*$ to satisfy (19).
      (Rule 2) Apply Rule 1 first and additionally perform
        $\mathbf{A}_{i,k} = *^c, \quad (\mathbf{A}_{i,k}, \mathbf{A}_{j,k}, \mathbf{A}_{add(i,j),k}) = (*,1,*)$ or $(*,*,1)$,
      where $*^c$ denotes the complement of $*$, i.e., $*^c = * \oplus 1$.
```