An Improved Greedy Cover Algorithm for Pliable Index Coding

***Abstract*—THIS PAPER IS ELIGIBLE FOR THE STUDENT PAPER AWARD. We present the ImpGrCov algorithm–a novel algorithm for devising binary linear codes for pliable index coding that have lower codelengths than those devised using state-of-the-art algorithms such as the greedy cover, and random cover algorithms. The proposed algorithm employs a novel, improved decoder that uses a representative matrix to keep track of relationships between undecoded messages, along with a demand-based exponential weighting of receivers. Simulations indicate 10-25% improvement in average codelength over the best performing algorithm in the literature.**

I. Introduction and Related Work

The index coding problem models a setup where a server communicates over a broadcast link with multiple clients having specific demands. In the general setting, the server is assumed to have multiple messages, and that each client already knows a subset of messages and demands a subset of messages that it does not already know. Though index coding focuses on a one-hop network, it has been shown to have a code-equivalence relationship with the more general network coding problem [1]–[3]. Due to this connection, there has been significant research interest in index coding and its variants [4]–[9].

One recent variant of index coding is the *pliable* index coding (PICOD(t)) problem where the demands of the clients are flexible in that the clients are satisfied as long as they receive *any* t new messages using the server's broadcast message and the messages they already have (prior to the receipt of the broadcast message). Thus, the server has more flexibility in selecting which t messages to convey to each client, and therefore more freedom in synthesizing the coded broadcast message. This feature of server flexibility in determining what to convey is natural in many real-world situations such as advertisement placement, and curated news content on websites and social media platforms.

In the introductory work by Brahma and Fragouli [10], it was established that quantifying the optimal broadcast rate (i.e., minimum number of transmissions) of PICOD problems is NP-hard by using a polynomial-time reduction from the monotone-1in3-SAT problem. In the same work, multiple algorithms to devise binary, linear codes for the PICOD problem, including the greedy cover (GrCov) and random cover (RandCov) algorithms, were proposed [10]. Using the latter algorithm, it was shown that the optimal broadcast rate of pliable index codes to convey t messages to n clients is at most $O(\min\{t\log^2 n, t\log n + \log^3 n\})$. Further, if the server only knows that each client has at least a messages and at most b messages, then the optimal broadcast rate among all linear codes was shown to be precisely $\min\{b+t, m-a\}$.

Focusing solely on linear codes, Song and Fragouli present the Binary Field Greedy (BinGreedy) algorithm with analytical performance guarantees [11]. Using this algorithm, it was shown that the optimal rate of pliable index coding is no more than $O(t\log n + \log^2 n)$. In [12] and [13], Liu and Tuninetti defined a class of complete PICOD(t) problems parameterized by a set S of integers, and present lower bounds on optimal broadcast rates using decoding chains and maximum acyclic induced subgraphs. Further, when S is a consecutive interval of integers (i.e., $S = \{a, a+1, \ldots, b\}$ for some positive integers a, b), or complement of a set of consecutive integers (i.e., $S = \{a, a+1, \ldots, b\}^c$), the optimal broadcast rate for a complete-S PICOD(t) instance with n clients among all pliable codes is $\min\{b+t, m-a\}$. Newer optimality results and lower bounds for optimal broadcast rate were established by Ong et al. by focusing on clients that are absent [14]. Recently, Krishnan et al. used conflict-free colorings of hypergraphs to devise general non-linear pliable codes, and build new connections between broadcast rate and variants of graph chromatic numbers [15].

This work presents the Improved Greedy Cover (ImpGrCov) algorithm to devise binary, linear code for PICOD(t) problems that incorporates three key ideas: (a) selection of maximal utility linear combinations similar to the GrCov algorithm; (b) exponential weighting of clients similar to the BinGreedy algorithm; and (c) an improved representative-based decoder that allows the server and the clients to efficiently keep track of linear combinations involving precisely two undecoded messages via a single representative matrix. While information from such linear combinations are ignored in GrCov, the representative matrix in ImpGrCov allows relations between messages to be learned to enable more messages to be decoded from fewer coded messages. Aided by these modifications, the ImpGrCov algorithm offers significant broadcast rate improvement at the same order of complexity. The paper is organized as follows. Section II presents notation and problem formulation. Section III presents the ImpGrCov algorithm, and lastly, Section IV presents the results of our simulations.

II. Preliminaries

1) Notation: In this work, matrices and vectors are denoted by bold font, e.g., $\boldsymbol{A}$, $\boldsymbol{B}$, etc. For a matrix $\boldsymbol{A}$, $\boldsymbol{A}_{i\bullet}$ and A_{ij} denote the i^{th} row and the $(i,j)^{\text{th}}$ entry of the matrix $\boldsymbol{A}$. For $n \in \mathbb{N}$, $[1:n] \triangleq \{1, 2, \ldots, n\}$. For a predicate P, $\mathbb{1}(P)$ is 1 if P is true, and 0 otherwise. For a tuple $\boldsymbol{X} = (X_1, \ldots, X_\ell)$ and index set $J \subseteq [1:\ell]$, we let $\boldsymbol{X}_J$ to be the ordered tuple with components of $\boldsymbol{X}$ specified by the indices in J. For example, if $\boldsymbol{X} = (X_1, X_2, X_3, X_4)$ and $J = \{1, 4\}$, then $\boldsymbol{X}_J = (X_1, X_4)$.

2) Problem Formulation: The setup of the Index Coding (IC) problem is that of a noiseless broadcast problem where a single server possessing m messages $M_1, M_2, \dots M_m \in \{0,1\}^L$ attempts to encode the messages jointly to communicate to n clients. Each client $j \in \{1, \dots, n\}$, is assumed to possess a subset of messages given by the index set $A_j \subsetneq [1:m]$ (i.e., client j knows $\{M_k : k \in A_j\}$), and specifically requests a particular message whose index is $w_j \notin A_j$ (i.e., client j wants M_{w_j}). The goal of the index coding problem is to characterize the smallest broadcast rate so that all clients are *satisfied*, i.e, each client decodes the message they request.

The setup of the *Pliable* Index Coding (PICOD(t)) problem is identical to the index coding problem except that the clients are satisfied with decoding *any* $t \in \mathbb{N}$ messages not already known. If the number of messages not known to client j is fewer than t, i.e., if $m - |A_j| < t$, then the client is satisfied if it decodes messages $\{M_k : k \in A_j^c\}$. Without loss of generality, we may assume that for any two clients $j \neq j'$, $A_j \neq A_{j'}$.

The information available to the clients is succinctly represented by a bipartite graph whose nodes are labeled by the m messages and n clients, and an edge exists between a client node j and a message node k if and only if client j does not know message M_j. In this work, $\boldsymbol{G}$ will represent the $n \times m$ biadjacency matrix of this graph representation of the PICOD problem; thus the PICOD(t) is fully specified by $(\boldsymbol{G}, t)$. Given the biadjacency matrix $\boldsymbol{G}$ of a PICOD problem, and a set $B \subseteq [1:m]$ of messages, we let $\mathcal{N}_1(B)$ to denote all clients that have exactly one message node in B as a neighbor, i.e., $\mathcal{N}_1(B) = \left\{ j \in [1:n] : \sum_{k \in B} G_{jk} = 1 \right\}$.

The server's aim is to efficiently and jointly code messages using a *pliable index code* that consists of the following:

- An *encoder* $E : \{\{0,1\}^L\}^m \to \{\{0,1\}^L\}^\ell$ that the server uses to encode messages[1] $\boldsymbol{M} = (M_1, M_2, \dots, M_m)$, and convey $E(\boldsymbol{M})$ to the n clients. Here, ℓ denotes the codelength;
- For each $j \in [1:n]$, t distinct indices $i_1(j), \dots, i_t(j) \in A_j^c$ denoting the messages that client j decodes; and
- For each client $j \in [1:n]$, a *decoding function*

$$D_j : \{\{0,1\}^L\}^\ell \times \{\{0,1\}^L\}^{|A_j|} \to \{\{0,1\}^L\}^t,$$

using which the client successfully decodes messages

$$(M_{i_1(j)}, \dots, M_{i_t(j)}) = D_j(E(\boldsymbol{M}), \boldsymbol{M}_{A_j}).$$

Note that if for a client $j \in [1:n]$, $m - |A_j| < t$, then the client's decoded message indices correspond to A_j^c, and the decoding function outputs $m - |A_j|$ message values instead.

The overall aims of the PICOD problem are to identify the optimal codelength for a given instance $(\boldsymbol{G}, t)$, and to devise codes with the optimal codelength. In this paper, we restrict our focus to a specific class of binary vector-linear pliable index codes, i.e., $E(M_1, \dots, M_m)$ is a collection of ℓ linear combinations of the form $\oplus_{k \in B} M_k$, where $\oplus$ denotes the component-wise XOR operation. We introduce a novel algorithm for devising a binary, vector-linear code for any PICOD(t) instance that offers significant codelength improvement over the state-of-the-art algorithms in the literature.

[1]Though we specify messages to be uniform random variables over $\{0,1\}^L$ for some L, the ideas apply even when the alphabet is any finite field.

III. Improved Algorithm for Pliable Index Coding

The Improved Greedy Cover (ImpGrCov) algorithm for pliable index coding combines three key ideas: (a) maximal greedy coded message selection similar to the GrCov algorithm [10]; (b) demand-based exponential weighting of clients from binary field greedy (BinGreedy) algorithm [11]; and (c) a novel representative-based decoding to offer improved performance at the same order of complexity (see Appendix A).

Given a PICOD problem $(\boldsymbol{G}, t)$, the proposed algorithm, similar to GrCov, iterates between two phases until all clients are satisfied: (a) selection of a maximal utility message (Lines 7-22 of Algorithm 3) by the encoder; and (b) sequential representative-based decoding, and graph and representative update by the encoder and clients (Lines 23-29 of Algorithm 3). A detailed description of the latter is presented first.

1) Client Operation: In the GrCov algorithm, upon receiving a coded message $\oplus_{j \in B} M_j$, each client *subtracts* the messages it already knows, and if what remains is a single message, the client then decodes a new message. Thus, only those clients in $\mathcal{N}_1(B)$ decode a message; the remaining clients discard the coded message. The edge connecting clients and their decoded messages are removed from $\boldsymbol{G}$, and this process is repeated until all clients decode the requisite number of messages.

In the ImpGrCov algorithm, client decoding and update operations are more nuanced due to the use of the representative matrix $\boldsymbol{H}$. At any point in the algorithm, the representative of a message k at client j is a message that client j *knows* is linearly related to k. For example, if $H_{j,k} = \ell$, then it must be the case that client j knows the value of $M_k \oplus M_\ell$, and hence, if either M_k or M_ℓ is decoded, then the other is decoded as well. At the beginning, the representative of each message index k at each client j is initialized with $H_{jk} = k$, i.e., each message is related only to itself. As coded messages are received, each client keeps track of any decoded linear combinations involving two messages. For example, if client j decodes $M_{i_1} \oplus M_{i_2}$ with $i_1 < i_2$, then by updating $H_{j,i_1} = H_{j,i_2} = \min\{i_1, i_2\}$, the client will be, at a later stage, able to decode both M_{i_1} and M_{i_2} from either. Thus, as the algorithm progresses, the relation between pairs of undecoded messages at each client is built and efficiently stored through the representative matrix $\boldsymbol{H}$, allowing for extra messages to be decoded with fewer transmissions.

During the ImpGrCov algorithm, upon receiving a coded message $\oplus_{k \in B} M_k$, each client $j \in [1:n]$ proceeds as follows.

- D1 It first identifies indices $B_j = B \cap \{k \in [1:m] : G_{jk} = 1\}$ of unknown messages that are encoded together. By subtracting known messages, the client computes $\oplus_{k \in B_j} M_k$.
- D2 It then uses $\boldsymbol{H}$ (at that time) to relate indices in B_j with their representatives, and compute $\oplus_{k \in B_j} M_{H_{jk}}$.
- D3 If $\oplus_{k \in B_j} M_{H_{jk}}$ involves only one message, say M_ℓ, then this message is decoded. Further, the client also decodes any other message whose representative is ℓ, i.e., any message M_k such that $H_{jk} = \ell$ is decoded.

Algorithm 1: SRD$(B, \boldsymbol{G}, \boldsymbol{H}, j, m)$

$\mathcal{J} \leftarrow \emptyset$
$B_j \leftarrow \{k \in B : G_{jk} = 1\}$
for $k \in [1 : m]$ **do**
  $T_k \leftarrow 0$
for $k \in B_j$ **do**
  $T_{H_{jk}} \leftarrow T_{H_{jk}} + 1$
$\mathcal{I} = \{k \in [1 : m] : T_k \text{ is odd}\}$
if $|\mathcal{I}| = 1$ **then**
  $i \leftarrow \min \mathcal{I}$
  $\mathcal{J} = \{k \in [1 : m] : H_{jk} = i\}$
return $\mathcal{J}$

Algorithm 2: RepUpdate$(B, \boldsymbol{G}, \boldsymbol{H}, n, m)$

for $j \in [1 : n]$ **do**
  $B_j \leftarrow \{k \in B : G_{jk} = 1\}$
  for $k \in [1 : m]$ **do**
    $T_{jk} \leftarrow 0$
  for $k \in B_j$ **do**
    $T_{jH_{jk}} \leftarrow T_{jH_{jk}} + 1$
  $\mathcal{I}_j = \{k \in [1 : m] : T_{jk} \text{ is odd}\}$
  if $|\mathcal{I}_j| = 2$ **then**
    $i_1 \leftarrow \min \mathcal{I}_j$
    $i_2 \leftarrow \max \mathcal{I}_j$
    $\mathcal{J} = \{k \in [1 : m] : H_{jk} = i_2\}$
    for $k \in \mathcal{J}$ **do**
      $H_{jk} = i_1$
return $\boldsymbol{H}$

D4 If $\oplus_{k \in B_j} M_{H_{jk}}$ involves two unknown messages, say $M_{\ell_1} \oplus M_{\ell_2}$ with $\ell_1 < \ell_2$, then the client identifies all messages whose representative is presently ℓ_2, and updates their representatives instead to be ℓ_1.

D5 Lastly, if $\oplus_{k \in B_j} M_{H_{jk}}$ involves three or more messages, the client discards the coded message.

Note that a client that decodes a message uses the Sequential Representative-based (SR) Decoder (Algorithm 1) to perform Steps D1, D2 and D3, and a client that updates its representatives uses the RepUpdate algorithm (Algorithm 2) to perform Steps D1, D2 and D4. The following example illustrates the client operations.

Example 1: Let $m = 5$, $n = 10$, and $t = 3$. Consider the perspective of, say, client 1 that only knows M_2. Initially, $\boldsymbol{G}_{1\bullet} = [1\ 0\ 1\ 1\ 1]$ and $\boldsymbol{H}_{1\bullet} = [1\ 2\ 3\ 4\ 5]$. Suppose the encoder sends $M_1 \oplus M_2 \oplus M_3 \oplus M_5$, $M_1 \oplus M_2 \oplus M_5$, $M_1 \oplus M_3 \oplus M_5$, and $M_1 \oplus M_3$, in that order. The client performs the following:

- On receiving the first coded message, client 1 subtracts M_2 to obtain $M_1 \oplus M_3 \oplus M_5$. Since $\boldsymbol{H}$ has no non-trivial representative relationships as yet, the client is unable to process the coded message and decode a message.
- Upon receiving the second coded message, this client subtracts M_2 to compute $M_1 \oplus M_5$; since this linear combination involves two messages, the client updates the representatives to $\boldsymbol{H}_{1\bullet} = [1\ 2\ 3\ 4\ \underline{1}]$.
- Upon receiving the third coded message, the client uses the relationship between messages and their representatives to decode $M_3 = M_1 \oplus M_3 \oplus M_1$. Hence, it sets $G_{1,3} = 0$.
- Lastly, upon receiving $M_1 \oplus M_3$, client 1 subtracts the recently decoded M_3 to identify M_1. Since 1 is the representative for 5 (i.e., it already knows $M_1 \oplus M_5$), it also decodes M_5. With this, client 1 becomes satisfied.

Remark 1: The performance of the sequential representative-based decoder depends on the order in which coded messages are conveyed. It is evident from Example 1 that client 1 decodes three messages if the encoder conveys $M_1 \oplus M_2 \oplus M_5$, $M_1 \oplus M_3 \oplus M_5$, and $M_1 \oplus M_3$ in that order. However, if the encoder instead conveyed $M_1 \oplus M_3 \oplus M_5$, $M_1 \oplus M_2 \oplus M_5$, and $M_1 \oplus M_3$, no messages will be decoded by the very same client.

2) Encoder Operation: The encoders of the ImpGrCov and GrCov algorithms share many similarities; both are greedy, myopic in coded message construction, and during each round generate a linearly coded message that has maximal utility. A critical difference lies in how the two algorithms define the *utility* of a coded message.

In the GrCov algorithm, the utility of a subset of message indices B is tied to $|\mathcal{N}_1(B)|$, whereas in ImpGrCov, we choose a different metric altogether. Suppose at any stage, each client j requires W_j additional messages to be satisfied. We introduce a measure of dissatisfaction given by $\sum_{j \in \mathbb{U}} 2^{W_j}$, where $\mathbb{U}$ represents all clients that still require messages to attain satisfaction. We then define the utility of a subset of message indices B to be the reduction in this measure of dissatisfaction $\sum_{j \in \mathbb{U}} 2^{W_j} - \sum_{j \in \mathbb{U}} 2^{W'_j}$, where W'_j denotes the number of messages client j still needs *after* the processing of $\oplus_{j \in B} M_j$, and is the difference between W_j and the number of messages that client j decodes using the sequential representative-based decoding upon receiving the $\oplus_{j \in B} M_j$. This measure assigns exponentially more weight to more starved clients decoding messages as opposed to less starved clients decoding the same number of messages. This allows the encoder to generate a coded message that assists more starved clients first. As in GrCov, greedy selection of a B with maximal utility is employed. Given $(\boldsymbol{G}, t)$, the ImpGrCov does the following:

E1 Each client j is given a weight W_j that is the minimum of t and the number of messages not known to the client.

E2 At each client, the representative of each message is set to itself.

E3 The code is initialized as the empty set, and since all clients are unsatisfied at the beginning, the set of unsatisfied clients $\mathbb{U} = [1 : n]$.

E4 The message indices B of the coded message to be transmitted is initialized as the empty set, and the utility δ of the coded message is set to zero.

E5 The encoder computes the measure of dissatisfaction $\sum_{j \in \mathbb{U}} 2^{W_j}$ to track the progress of the algorithm.

E6 For each message k in B^c and client j, the encoder determines the messages $S_{j,k}$, and the number $\ell_{j,k}$ of messages that client j decodes (using the SR decoder) if

messages with indices in $B \cup \{k\}$ are coded together.

E7 For each message k in B^c, the encoder computes

$$\Delta_k = \sum_{j \in \mathbb{U}} \left(2^{W_j} - 2^{W_j - \ell_{j,k}}\right),$$

which is viewed as the (incremental) utility of message k in enabling the satisfaction of all clients.

E8 If there is a k in B^c whose utility $\Delta_k > \delta$, then it is possible to improve the utility of the present message set B. In this case, the encoder picks a message index k^* that maximizes the utility. If there are multiple choices, ties are broken randomly. The chosen index k^* is added to B, δ is updated to Δ_{k^*}, and the algorithm returns to Step E6.

E9 On the other hand, if no k offers strictly greater utility, then the identified index set B is maximal. Therefore, B is added to the code, and $\oplus_{j \in B} M_j$ is transmitted.

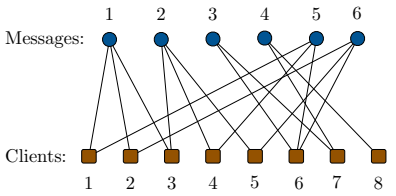
E10 The coded message $\oplus_{j \in B} M_j$ is then used to decode messages, update $\boldsymbol{G}$ and the representative matrix $\boldsymbol{H}$.

E11 Using the k^* (chosen in Step E8), the weights of each client j is updated to $W_j - \ell_{j,k^*}$, and the unsatisfied client set $\mathbb{U}$ is updated to be those clients with positive updated weights. The algorithm halts if $\mathbb{U}$ is empty, else, it reverts to Step E4 to select another coded message.

The following example illustrates the encoder operations.

Example 2: Consider the following PICOD instance with $m = 6$ messages, $n = 8$ clients, and $t = 3$.

$$\boldsymbol{G} \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$



The encoder begins by setting $\boldsymbol{W} = [2, 2, 2, 2, 2, 3, 2, 1]$. Note that every client other than client 6 is assigned a weight of 2 or less since they already know 4 or more messages. Next, the representative matrix is initialized with

$$\boldsymbol{H} \triangleq \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix},$$

since the representative of each message at every client is itself. The code is initially empty, and all clients are unsatisfied.

To generate the first coded message, the encoder sets $B = \emptyset$, $\delta = 0$, and scans through all message indices $k \notin B$ to determine their utility by identifying the set $S_{j,k}$ of indices of messages decoded by client j assuming messages with indices in $B \cup \{k\}$ were coded together. Thus, for $j \in [1:n]$, and $k \in [1:m]$,

$$S_{j,k} = \begin{cases} \{k\} & G_{jk} = 1 \\ \emptyset & \text{otherwise} \end{cases}, \text{ and } \ell_{j,k} = G_{j,k}.$$

Using this, the utility of the 6 messages are computed to be $\boldsymbol{\Delta} = [6, 6, 6, 3, \underline{8}, \underline{8}]$. Hence, the encoder identifies M_5 and M_6 to be most useful. Note that even though the column weights of the third, fifth and sixth columns are 3, the third is not deemed as useful as the other two due to the exponential weighting. Let us suppose $k^* = 5$ is chosen at random to build the coded message. The encoder updates $B = \{5\}$ and sets $\delta = \Delta_5 = 8$.

It now attempts to add another message to add to B. Upon scanning through $k \notin B = \{5\}$, the encoder re-computes their utilities to be $\boldsymbol{\Delta} = [10, 10, 6, \underline{11}, *, 8]$. Thus, it now identifies $k^* = 4$ as the unique choice that maximizes utility. The encoder then updates $B = \{4, 5\}$ and $\delta = \Delta_4 = 11$. Next round of scanning for $k \notin \{4, 5\}$ yields $\boldsymbol{\Delta} = [\underline{13}, \underline{13}, 9, *, *, 11]$ as the utilities. The encoder has two equally good choices for k^*, and suppose it chooses $k^* = 2$. Then, it updates $B = \{2, 4, 5\}$ and $\delta = \Delta_2 = 13$. Computing the utilities again yields $\boldsymbol{\Delta} = [11, *, 7, *, *, 9]$. Since there is no strictly improving choice, the encoder determines $B = \{2, 4, 5\}$ to be a maximal utility set. The encoder now updates $C = \{\{2, 4, 5\}\}$, allowing clients $1, 3, 5, 6, 7$, and 8 to decode message 5, 2, 2, 5, 4, and 4, respectively. Thus, weights are updated to $\boldsymbol{W} = [\underline{1}, 2, \underline{1}, 2, \underline{1}, \underline{2}, \underline{1}, \underline{0}]$, and $\boldsymbol{G}$ is updated as

$$G_{1,5} = G_{3,2} = G_{5,2} = G_{6,5} = G_{7,4} = G_{8,4} = 0.$$

At this point, the encoder updates the representative matrix. Since client 4 can compute $M_2 \oplus M_5$, client 4's representative vector is updated to be $H_{4\bullet} = [1, 2, 3, 4, \underline{2}, 6]$. Since $\mathbb{U}$ is now $[1:5]$, the search for another coded message starts with $B = \emptyset$

Algorithm 3: ImpGrCov$(\boldsymbol{G}, n, m, t)$

```
for j ∈ [1 : n] do
    W_j ← min{t, Σ_{k∈[1:m]} G_jk}
    for l ∈ [1 : m] do
        H_jl ← l
C ← ∅
U ← [1 : n]
while U ≠ ∅ do
    B ← ∅
    δ ← 0
    Flag ← 1
    while Flag do
        for k ∉ B do
            for j ∈ U do
                S_{j,k} ← SRD(B ∪ {k}, G, H, j, m)
                ℓ_{j,k} ← min{|S_{j,k}|, W_j}
            Δ_k ← Σ_{j∈U} 2^{W_j} (1 − 2^{−ℓ_{j,k}})
        if (max_{k∉B} Δ_k > δ) then
            k* ← arg max_{k∉B} Δ_k
            δ ← Δ_{k*}
            B ← B ∪ {k*}
        else
            Flag ← 0
    C = C ∪ {B}
    for j ∈ U do
        W_j ← W_j − ℓ_{j,k*}
    U ← {j ∈ [1 : n] : W_j > 0}
    for j ∈ [1 : n] and l ∈ S_{j,k*} do
        G_jl ← 0
    H ← RepUpdate(B, G, H, n, m)
return C
```

TABLE I: Comparison of BinGreeedy (BG), RandCov (RC), Greedy Cover (GC) and ImpGrCov Algorithms

$n = m = 100,\ p = 0.2$						$n = m = 100,\ p = 0.5$					$n = m = 100,\ p = 0.8$				
t	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease
1	7.59	10.27	3.17	3.15	0.84%	3.95	8.61	3.78	3.8	-0.36%	2	5.9	2	2	0%
3	18.17	19.34	7.77	6.97	10.32%	8.8	16.04	8.25	7.75	5.99%	5	11.4	5.3	5.02	5.31%
5	28.83	27.29	11.71	10.09	13.76%	13.58	22.65	12.31	11.09	9.87%	8.1	16.28	8.13	7.56	6.99%
10	56.66	47.23	21.25	17.82	16.17%	24.72	37.81	21.9	19.08	12.85%	14.89	27.78	15.03	13.94	7.25%
15	83.33	65.28	30.93	25.05	19.02%	35.59	52.42	30.94	26.68	13.75%	21.41	38.86	21.71	20.11	7.4%
20	94.95	76.14	40.17	29.83	25.72%	46.7	66.9	39.97	34.15	14.57%	27.93	49.66	28.39	26.4	7.01%
$n = m = 200,\ p = 0.2$						$n = m = 200,\ p = 0.5$					$n = m = 200,\ p = 0.8$				
t	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease
1	9.04	12.18	3.99	3.99	-0.05%	4.34	9.8	4.02	4.03	-0.07%	2.48	6.52	2.48	2.48	-0.08%
3	20.27	22.01	8.95	8.01	10.5%	9.71	17.67	9.3	8.73	6.07%	5.78	12.5	5.74	5.29	7.81%
5	31.27	30.46	13.38	11.62	13.13%	14.73	24.46	13.78	12.36	10.28%	8.75	17.54	8.72	8.05	7.69%
10	58.45	50.1	23.4	19.77	15.52%	25.99	40	23.99	20.82	13.2%	15.79	29.33	15.7	14.4	8.27%
15	86.1	69.06	32.89	27.38	16.76%	36.98	54.69	33.67	28.81	14.43%	22.52	40.86	22.51	20.84	7.4%
20	115.16	88.64	42.08	34.98	16.88%	47.73	68.82	43.25	36.65	15.26%	29.1	51.73	29.24	27.04	7.5%
$n = m = 500,\ p = 0.2$						$n = m = 500,\ p = 0.5$					$n = m = 500,\ p = 0.8$				
t	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease	BG	RC	GC	IGC	Decrease
5	34.74	32.94	15.74	13.81	12.28%	16.35	26.74	15.55	13.98	10.08%	9.38	19.14	9.48	8.91	5.94%
10	62.64	52.4	27.43	22.97	16.26%	28.11	42.9	26.66	23.02	13.65%	16.71	31.31	16.74	15.26	8.84%
15	90.01	70.79	38.48	31.47	18.21%	39.29	57.89	37.22	31.7	14.81%	23.66	43.12	23.7	21.85	7.82%
20	117.21	88.36	49.07	39.67	19.15%	50.17	72.43	47.51	40.16	15.46%	30.43	54.41	30.55	28.06	8.15%

and $\delta = 0$. As before, for each $k \notin B = \emptyset$, and $j \in [1:8]$, the encoder identifies the set $S_{j,k}$ of indices of messages decoded by client j if messages with indices $B \cup \{k\}$ were coded together. Doing so yields

$$S_{j,k} = \begin{cases} \{1\} & (j,k) = (1,1), (2,1), \text{ or } (3,1) \\ \{2,5\} & (j,k) = (4,2) \text{ or } (4,5) \\ \{3\} & (j,k) = (6,3) \text{ or } (7,3) \\ \{6\} & (j,k) = (2,6), (5,6) \text{ or } (6,6) \\ \emptyset & \text{otherwise} \end{cases}.$$

Conveying M_2 or M_5 uncoded allows client 4 to decode two messages since it already knows $M_2 \oplus M_5$ (by recording the representatives in $\boldsymbol{H}$). Using this, the utility of the messages are computed to be $\boldsymbol{\Delta} = [\underline{5}, 4, 3, 0, \underline{5}, \underline{5}]$. The encoder now has three choices for k^*, and suppose it picks $x^* = 6$. Then, it sets $B = \{6\}$ and $\delta = \Delta_6 = 5$. In the next round, the encoder computes $\boldsymbol{\Delta} = [6, 8, 4, 5, \underline{10}, *]$, and B is updated to be $\{5, 6\}$ and $\delta = \Delta_5 = 10$. A third round results in $\boldsymbol{\Delta} = [6, 7, 9, \underline{10}, *, *]$, revealing $B = \{5, 6\}$ as a maximal utility set. The encoder now updates $C = \{\{2, 4, 5\}, \{5, 6\}\}$, $\boldsymbol{W} = [1, \underline{1}, 1, \underline{0}, \underline{0}, \underline{1}, 1, 0]$, and

$$G_{2,6} = G_{4,2} = G_{4,5} = G_{5,6} = G_{6,6} = 0.$$

There is no update to the representative matrix in this round, and clients 4, 5 and 8 are satisfied. A third coded message selection then yields a unique maximal choice of $B = \{1, 3\}$ that satisfies the remaining clients. Thus, $M_2 \oplus M_4 \oplus M_5$, $M_5 \oplus M_6$, and $M_1 \oplus M_6$ is the pliable code that ImpGrCov yields.

IV. Simulation Results

We now present simulation results that compare the ImpGrCov algorithm with GrCov [10], BinGreedy [11], and RandCov (with $r = 3$) algorithms [10]. We consider *homogeneous* PICOD instances with $m = n$ where $\boldsymbol{G}$ is a random binary $n \times m$ matrix with entries that are independent and identically distributed according to the Bernoulli distribution with parameter p. Simulations were performed for $p \in \{0.2, 0.5, 0.8\}$ (signifying high, medium, and low *a priori* side-information settings), for $n \in \{100, 200, 500\}$, and for varying values of t. For $n = 100$ and 200, $t = 1, 3, 5, 10, 15$, and 20 were simulated, and for $n = 500$, $t = 5, 10, 15$, and 20 were simulated.

For each $(n = m, p)$-pair, 1000 instances of homogeneous graphs were randomly generated, and the codelengths of codes from the four algorithms were recorded. Table I reports the average codelengths for the simulated $(n = m, p, t)$ settings, and corresponding figures can be found in Appendix B. The table also reports the percentage decrease in average codelength of codes from proposed ImpGrCov algorithm over those from GrCov, since the latter outperforms those from BinGreedy and RandCov algorithms. The following remarks are in order.

- **For a fixed $(n = m, p)$ pair, the improvement offered by ImpGrCov generally increases with t.** For $t = 1$, there is little benefit. Note that for $t = 1$ setting, the exponential weighting plays no role since the weights associated with the clients are either zero or one. However for larger t's, ImpGrCov offers $8 - 25\%$ improvement.
- **For a fixed $(n = m, t)$ pair with $t > 1$, the improvement offered by ImpGrCov generally decreases with p.** The improvement for $p = 0.2$ ranges from 10-25% whereas that for $p = 0.8$ hovers around 7.5%. This can be attributed to the fact that when the clients have more side information, there is a greater chance for clients to be able to update their representatives, which then translates to a greater chance of decoding additional messages from future coded messages.

To understand the effect of exponential weighting in Line 16 of Algorithm 3, we simulated for various homogeneous PICOD settings an unweighted variant of the ImpGrCov algorithm where the line was replaced by $\Delta_k \leftarrow \sum_{j \in \mathbb{U}} \ell_{j,k}$. The performance of this variant was noted to be generally better than that of GrCov, but worse than ImpGrCov, indicating that the exponential weighting indeed helps with codelength reduction. The details of this comparison can be found in Appendix B.

REFERENCES

[1] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3187–3195, July 2010.

[2] M. Effros, S. El Rouayheb, M. Langberg, "An equivalence between network coding and index coding," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2478–2487, May 2015.

[3] L. Ong, J. Kliewer, B. N. Vellambi, and P. L. Yeoh, "A code equivalence between secure network and index coding," *IEEE Journal on Selected Areas in Inf. Theory*, vol. 2, no. 1, pp. 106-120, Mar. 2021.

[4] F. Arbabjolfaei and Y.-H. Kim, "Fundamentals of index coding," *Foundations and Trends in Commun. and Inf. Theory*, vol. 14 no. 3–4, pp. 163–346, 2018.

[5] Z. Bar-Yossef, Y. Birk, T. S. Jayram, T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, Mar. 2011.

[6] S. H. Dau, V. Skachek, Y. M. Chee, "On the security of index coding with side information," *IEEE Trans. Inf. Theory*, vol. 58, no. 6, pp. 3975–3988, June 2012.

[7] L. Ong, B. N. Vellambi, P. L. Yeoh, J. Kliewer, and J. Yuan, "Secure index coding: Existence and construction," in *Proc. IEEE Int. Symp. Inf. Theory*, July 2016, pp. 2834–2838.

[8] M. J. Neely, A. S. Tehrani, Z. Zhang, "Dynamic index coding for wireless broadcast networks," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, Nov. 2013.

[9] L. Ong, C. K. Ho, and F. Lim, "The single-uniprior index-coding problem: The single-sender case and the multi-sender extension," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 3165–3182, June 2016.

[10] S. Brahma and C. Fragouli, "Pliable index coding," *IEEE Trans. Inf. Theory*, vol. 60, no. 11, pp. 6192–6203, Nov. 2014.

[11] L. Song and C. Fragouli, "A polynomial-time algorithm for pliable index coding," *IEEE Trans. Inf. Theory*, vol. 64, no. 2, pp. 979–999, Feb. 2018.

[12] T. Liu and D. Tuninetti, "Information theoretic converse proofs for some PICOD problems," in *Proc. IEEE Inf. Theory Workshop*, Kaohsiung, Taiwan, Nov. 6–10, 2017, pp. 284–288.

[13] T. Liu and D. Tuninetti, "An information theoretic converse for the "consecutive complete-S" PICOD problem," in *Proc. IEEE Inf. Theory Workshop*, Guangzhou, China, Nov. 25–29, 2018, pp. 165–169.

[14] L. Ong, B. N. Vellambi, J. Kliewer and P. Sadeghi, "Improved lower bounds for pliable index coding using absent receivers," in Proc. Int. Zurich Seminar on Inf. and Commun., Zurich, Switzerland, Feb 2020, pp. 26–30.

[15] P. Krishnan, R. Mathew and S. Kalyanasundaram, "Pliable index coding via conflict-free colorings of hypergraphs," in *Proc. IEEE Int. Symp. Inf. Theory*, July 2021, pp. 214-219

Appendix A
Computational Complexity of ImpGrCov and GrCov

For the sake of comparison, we provide the greedy cover algorithm here (Algorithm 4). Note that the structure of GrCov and ImpGrCov algorithms are identical. The difference lies in: (a) the computation of the utility of a message; (b) in determining the maximal utility coded message; and (c) in the post-processing after message selection.

Given a PICOD instance $(\boldsymbol{G}, t)$ the complexity of Algorithm 4 can be estimated as follows:

- The outermost while loop (Lines 5-23) corresponds to repeating the coded message selection routine to satisfy clients. Each round delivers at least one message to a client, and hence, a maximum of $O(nt)$ rounds of this loop will be attempted for any PICOD instance.
- The next while loop (Lines 9-17) attempts to identify a maximal utility message index set B. Each round adds a message to B, and hence, this loop is attempted at most $O(m)$ times in each round of the outermost loop.
- Each round of the innermost for loop (Lines 10-11) requires computing row sum of a matrix of dimension $n \times (|B|+1)$. The columns corresponds to indices in B and one index from B^c. A naïve implementation of Lines 10-11 will compute the row sum corresponding to columns of B several times. This can be economized by first computing the row sums of the columns corresponding to B, and then for each round of the for loop, process the row-sum result with the additional column corresponding to $k \notin B$ to compute Δ_k. This computes all Δ's at a complexity of $O(mn)$.
- Lastly, the complexity of Lines 12-17, Lines 19-20, Line 21, and Lines 22-23 can be easily seen to be $O(m)$, $O(n)$, $O(n)$, and $O(mn)$, respectively.

Piecing together, we see that the computational complexity of the GrCov algorithm is $O(m^2n^2t)$. Similarly, the complexity of Algorithm 3 can be estimated as follows:

- The outermost while loop (Line 7-29) corresponds to repeating the coded message selection routine to satisfy clients. Each round delivers at least one message to a client, and hence, a maximum of $O(nt)$ rounds of this loop will be attempted for any PICOD instance.
- The next while loop (Lines 11-22) attempts to identify a maximal utility message index set B. Each round adds a message to B, and hence, this loop is attempted at most $O(m)$ times in each round of the outermost loop.
- The for loop (Lines 12-16) identifies for every $k \in B^c$ the messages that each client can decode when the message indices corresponding to $B \cup \{k\}$ are coded together. As in the GrCov discussion, a naïve implementation is inefficient. The very same computational task can be accomplished by first identifying what each client will decode assuming $\oplus_{j\in B} M_j$ is transmitted. This can be computed by n calls to Algorithm 1, which in total has $O(mn)$ complexity. Once this is done, for each $k \in B^c$, we can incrementally compute the messages each client decodes from $\left(\oplus_{j\in B} M_j\right) \oplus M_k$ using the SR decoder in $O(n)$ complexity. Thus, the overall complexity of all the computations in Lines 12-16 is then only $O(mn)$.
- Lastly, the complexity of Lines 17-22, Lines 24-25, Line 26, Lines 27-28, and Lines 29 can be verified to be $O(m)$, $O(n)$, $O(n)$, $O(mn)$ and $O(mn)$, respectively.

Again, piecing together, we see that the computational complexity of the ImpGrCov algorithm is also $O(m^2n^2t)$.

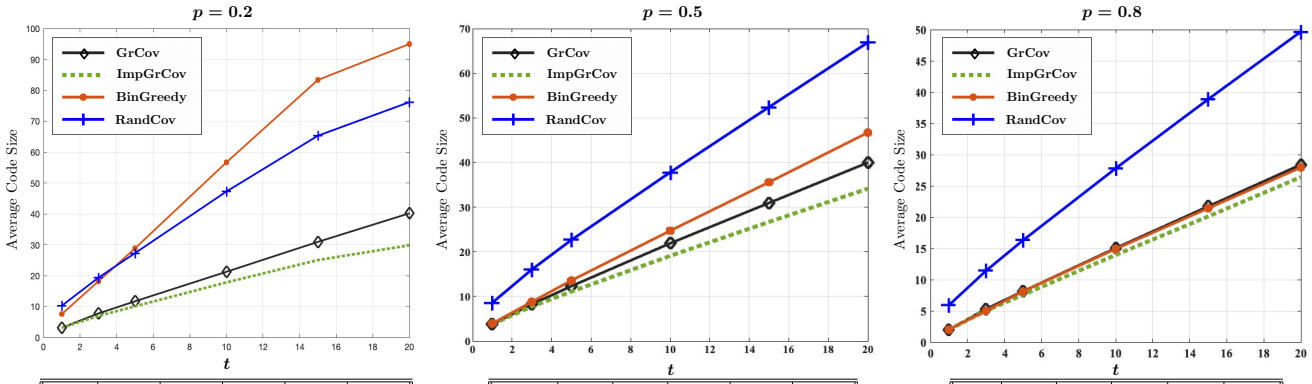
Algorithm 4: GrCov$(\boldsymbol{G}, n, m, t)$

1 **for** $j \in [1:n]$ **do**
2 $\quad W_j \leftarrow \min\left\{t, \sum_{k\in[1:m]} G_{jk}\right\}$
3 $C \leftarrow \emptyset$
4 $\mathbb{U} \leftarrow [1:n]$
5 **while** $\mathbb{U} \neq \emptyset$ **do**
6 $\quad B \leftarrow \emptyset$
7 $\quad \delta \leftarrow 0$
8 $\quad$ `Flag` $\leftarrow 1$
9 $\quad$ **while** `Flag` **do**
10 $\quad\quad$ **for** $k \notin B$ **do**
11 $\quad\quad\quad \Delta_k \leftarrow |\mathcal{N}_1(B \cup \{k\})|$
12 $\quad\quad$ **if** $\left(\max_{k\notin B} \Delta_k > \delta\right)$ **then**
13 $\quad\quad\quad k^* \leftarrow \arg\max_{k\notin B} \Delta_k$
14 $\quad\quad\quad \delta \leftarrow \Delta_{k^*}$
15 $\quad\quad\quad B \leftarrow B \cup \{k^*\}$
16 $\quad\quad$ **else**
17 $\quad\quad\quad$ `Flag` $\leftarrow 0$
18 $\quad C = C \cup \{B\}$
19 $\quad$ **for** $j \in \mathbb{U}$ **do**
20 $\quad\quad W_j \leftarrow W_j - \ell_{j,k^*}$
21 $\quad \mathbb{U} \leftarrow \{j \in [1:n] : W_j > 0\}$
22 $\quad$ **for** $j \in B$ *and* $i \in \mathcal{N}_1(B)$ **do**
23 $\quad\quad \boldsymbol{G}_{ij} \leftarrow 0$
24 **return** C

Appendix B
Simulation Results

Figures 1, 2, and 3 present the simulation data in Table I, and allow us to compare and judge the performance of the algorithms visually.

Table II presents the average codelengths obtained from the greedy cover (GrCov), improved greedy cover (ImpGrCov), and the unweighted variant of the greedy cover where Line 16 of Algorithm 3 is replaced by $\Delta_k \leftarrow \sum_{j\in\mathbb{U}} \ell_{j,k}$. Thus, in this variant, the utility of a coded message is given by the total number of messages the clients will decode when processing that coded message. From the table, it can be observed that when $t > 1$, the average codelengths of the unweighted variant of ImpGrCov are better than those of GrCov, but worse than those of ImpGrCov. For $p = 0.8$, note that there is insignificant difference in average codelengths offered by the unweighted variant of the ImpGrCov and GrCov algorithms.

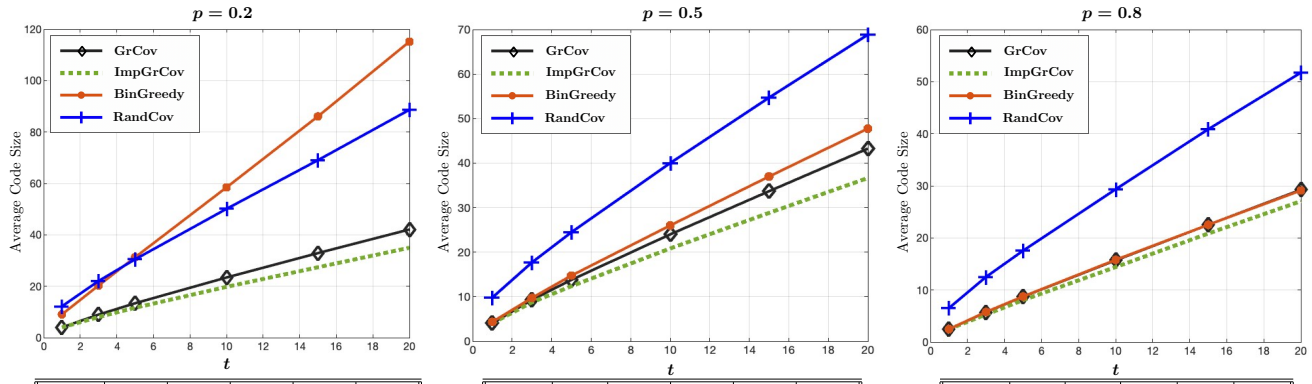


$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
0.85%	10.33%	13.77%	16.17%	19.02%	25.73%

$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
−0.36%	6.00%	9.87%	12.85%	13.75%	14.58%

$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
0%	5.32%	7.00%	7.26%	7.41%	7.01%

Fig. 1: Comparison of PICOD algorithms for $n = m = 100$ for varying t and p.

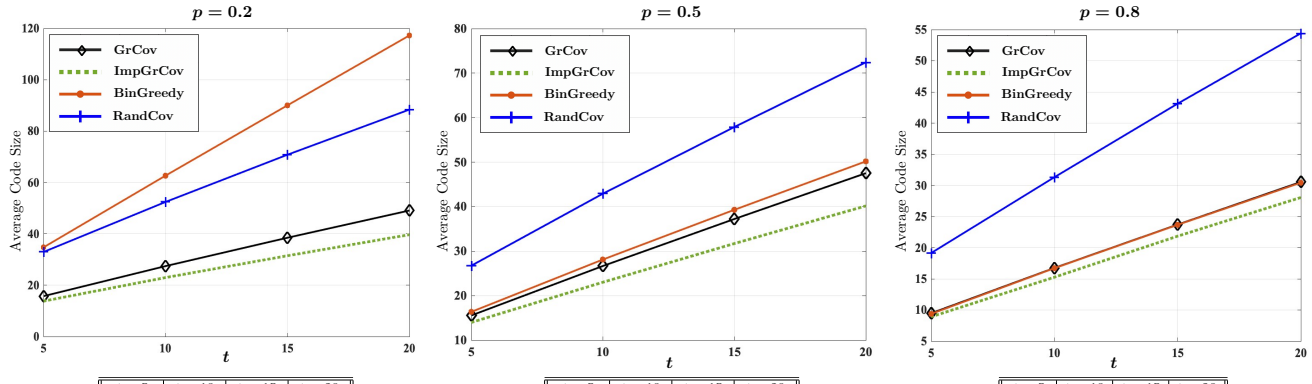


$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
−0.05%	10.50%	13.14%	15.52%	16.76%	16.89%

$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
−0.07%	6.08%	10.28%	13.20%	14.43%	15.26%

$t=1$	$t=3$	$t=5$	$t=10$	$t=15$	$t=20$
−0.09%	7.82%	7.70%	8.27%	7.40%	7.51%

Fig. 2: Comparison of PICOD algorithms for $n = m = 200$ for varying t and p.



$t=5$	$t=10$	$t=15$	$t=20$
12.29%	16.27%	18.21%	19.16%

$t=5$	$t=10$	$t=15$	$t=20$
10.08%	13.66%	14.81%	15.47%

$t=5$	$t=10$	$t=15$	$t=20$
5.95%	8.85%	7.82%	8.16%

Fig. 3: Comparison of PICOD algorithms for $n = m = 500$ for varying t and p.

TABLE II: Average Codelengths for GrCov, Unweighted ImpGrCov and ImpGrCov Algorithms

$n = m = 100,\ p = 0.2$				$n = m = 100,\ p = 0.5$			$n = m = 100,\ p = 0.8$		
t	GrCov	Unweighted ImpGrCov	ImpGrCov	GrCov	Unweighted ImpGrCov	ImpGrCov	GrCov	Unweighted ImpGrCov	ImpGrCov
1	3.17	3.13	3.15	3.78	3.79	3.8	2	2	2
3	7.77	7.19	6.97	8.25	7.94	7.75	5.3	5.31	5.02
5	11.71	10.59	10.09	12.31	11.45	11.09	8.13	8.13	7.56
10	21.26	18.91	17.82	21.90	19.74	19.09	15.03	15.02	13.94
$n = m = 200,\ p = 0.2$				$n = m = 200,\ p = 0.5$			$n = m = 200,\ p = 0.8$		
t	GrCov	Unweighted ImpGrCov	ImpGrCov	GrCov	Unweighted ImpGrCov	ImpGrCov	GrCov	Unweighted ImpGrCov	ImpGrCov
1	3.99	3.99	3.99	4.02	4.04	4.03	2.48	2.49	2.48
3	8.95	8.33	8.01	9.3	9.01	8.73	5.74	5.75	5.29
5	13.38	12.11	11.62	13.78	12.87	12.36	8.72	8.74	8.05
10	23.41	20.74	19.78	24.00	21.85	20.83	15.70	15.69	14.41