Concentrated Stopping Set Design for Coded Merkle Tree: Improving Security Against Data Availability Attacks in Blockchain Systems

Debarnab Mitra, Lev Tauz, and Lara Dolecek

Electrical and Computer Engineering
University of California, Los Angeles

Non-Volatile Memories Workshop
May 10, 2022
Table of Contents

1. Blockchain Background
2. Data Availability Attacks
3. LDPC Code Design
4. Conclusion
Blockchain

- Institutional Trust Systems
Blockchain

- Institutional Trust Systems
Blockchain Background

Blockchain

- Institutional Trust Systems
  - All parties trust an established institution
Blockchain

- Institutional Trust Systems
  - All parties trust an established institution

- Distributed/Decentralized Trust Systems
Blockchain

- **Institutional Trust Systems**
  - All parties trust an established institution

- **Distributed/Decentralized Trust Systems**
  - Multiple parties collaborate on a specific task without parties trusting one another
Blockchain Background

Blockchain

- Institutional Trust Systems
  - All parties trust an established institution

- Distributed/Decentralized Trust Systems
  - Multiple parties collaborate on a specific task without parties trusting one another

- Blockchain:
Blockchain

▶ Institutional Trust Systems
  - All parties trust an established institution

▶ Distributed/Decentralized Trust Systems
  - Multiple parties collaborate on a specific task without parties trusting one another

☐ Blockchain: Allows for decentralized trust systems
Blockchain

▶ Institutional Trust Systems
  • All parties trust an established institution

▶ Distributed/Decentralized Trust Systems
  • Multiple parties collaborate on a specific task without parties trusting one another

Blockchain: Allows for decentralized trust systems
  • Main application of Blockchain: Currency and Finance
Blockchain Ledger

<table>
<thead>
<tr>
<th>Blockchain Ledger</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A deposits $100</td>
</tr>
<tr>
<td>2. B deposits $150</td>
</tr>
<tr>
<td>3. C deposits $50</td>
</tr>
<tr>
<td>4. D deposits $200</td>
</tr>
<tr>
<td>5. A pays $20 to B</td>
</tr>
<tr>
<td>6. B pays $30 to C</td>
</tr>
</tbody>
</table>

... 

35. C pays $10 to D  
36. D pays $30 to A  
37. B pays $20 to D  
38. C pays $10 to B

Ledger of transactions
### Blockchain Ledger

- **Ledger of transactions**
- **Arranged in the form of blocks**

<table>
<thead>
<tr>
<th>Blockchain Ledger</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A deposits $100</td>
</tr>
<tr>
<td>2. B deposits $150</td>
</tr>
<tr>
<td>3. C deposits $50</td>
</tr>
<tr>
<td>4. D deposits $200</td>
</tr>
<tr>
<td>5. A pays $20 to B</td>
</tr>
<tr>
<td>6. B pays $30 to C</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>35. C pays $10 to D</td>
</tr>
<tr>
<td>36. D pays $30 to A</td>
</tr>
<tr>
<td>37. B pays $20 to D</td>
</tr>
<tr>
<td>38. C pays $10 to B</td>
</tr>
</tbody>
</table>
Blockchain Ledger

- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
Blockchain Ledger

- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Nodes store the ledger in their memory
Blockchain Ledger

- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Nodes store the ledger in their memory
- Operate on the ledger to validate transactions
Blockchain Ledger

- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Nodes store the ledger in their memory
- Operate on the ledger to validate transactions

NVM technologies like persistent memories: low latency, high reliability
Blockchain Ledger

- Ledger of transactions
- Arranged in the form of blocks
- Stored by a network of nodes
- Nodes store the ledger in their memory
- Operate on the ledger to validate transactions

NVM technologies like persistent memories: low latency, high reliability

⇒ Faster blockchains
Central Problem: Prohibitive Storage Overhead

▶ Each node stores a copy of the ledger in its memory
Central Problem: Prohibitive Storage Overhead

- Each node stores a copy of the ledger in its memory

Significant storage overhead
Central Problem: Prohibitive Storage Overhead

- Each node stores a copy of the ledger in its memory

Significant storage overhead

- Bitcoin ledger size $\sim 300\text{GB}^1$
- Ethereum ledger size $\sim 650\text{GB}^2$

As of 4/28/2022, $^1$https://www.blockchain.com/charts/blocks-size
$^2$https://etherscan.io/chartsync/chaindefault
Central Problem: Prohibitive Storage Overhead

Each node stores a copy of the ledger in its memory

Prohibitive for resource limited nodes

Significant storage overhead

- Bitcoin ledger size $\sim 300\text{GB}^1$
- Ethereum ledger size $\sim 650\text{GB}^2$

As of 4/28/2022,

$^1$https://www.blockchain.com/charts/blocks-size

$^2$https://etherscan.io/chartsync/chaindefault
Allowing Light Nodes

Light Nodes:

- Only store block headers (total size \(\sim 1GB\) for Ethereum)
- Can verify transaction inclusion in a block
- Cannot verify transaction correctness
  - Rely on honest Full nodes for fraud notification
  - Full nodes send verifiable fraud proofs to the light nodes to reject invalid blocks

Mitra, Tauz, Dolecek (UCLA)
Allowing Light Nodes

Light Nodes:

▶ Only store block headers (total size ~ 1GB for Ethereum)
Allowing Light Nodes

Light Nodes:
- Only store block headers (total size $\sim 1$GB for Ethereum)
- Can verify transaction inclusion in a block

Mitra, Tauz, Dolecek (UCLA)
Allowing Light Nodes

Light Nodes:
- Only store block headers
  (total size ~ 1GB for Ethereum)
- Can verify transaction inclusion in a block
- Cannot verify transaction correctness

Mitra, Tauz, Dolecek (UCLA)
Allowing Light Nodes

Light Nodes:

- Only store block headers (total size $\sim 1\text{GB}$ for Ethereum)
- Can verify transaction inclusion in a block
- Cannot verify transaction correctness → Rely on honest Full nodes for fraud notification
Allowing Light Nodes

Light Nodes:

- Only store block headers (total size $\sim 1$GB for Ethereum)
- Can verify transaction inclusion in a block
- Cannot verify transaction correctness → Rely on honest Full nodes for fraud notification
- Full nodes send verifiable fraud proofs to the light nodes to reject invalid blocks
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

Adversary creates an invalid block

Header Invalid Transactions
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

Adversary creates an invalid block

Adversary: Provides block to Full node but hides invalid portion

Honest Nodes: Cannot verify missing transactions
→ No fraud proof

Light Nodes: No fraud proof
→ Accept the header
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

Adversary creates an invalid block

▶ Adversary: Provides block to Full node but hides invalid portion
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

▶ Adversary: Provides block to Full node but hides invalid portion
Provides header to Light node
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

- **Adversary**: Provides block to Full node but hides invalid portion
  - Provides header to Light node

- **Honest Nodes**: Cannot verify missing transactions
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

- Adversary: Provides block to Full node but hides invalid portion
  - Provides header to Light node
- Honest Nodes: Cannot verify missing transactions → No fraud proof
Data Availability (DA) Attack

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

- Adversary: Provides block to Full node but hides invalid portion
- Provides header to Light node
- Honest Nodes: Cannot verify missing transactions → No fraud proof
- Light Nodes: No fraud proof
**Data Availability (DA) Attack**

Systems with light nodes and a dishonest majority of full nodes are vulnerable to DA attacks [Al-Bassam ’18], [Yu ’19]

- Adversary creates an invalid block
- Adversary: Provides block to Full node but hides invalid portion
  - Provides header to Light node
- Honest Nodes: Cannot verify missing transactions $\rightarrow$ No fraud proof
- Light Nodes: No fraud proof $\rightarrow$ Accept the header
Ensuring Data Availability

- Adversary
- Light Node

Mathematical expressions:

- For Probability of failure using 2 random samples:
  \[1 - \frac{1}{16} = 0.87\]
  \[1 - \frac{1}{15} = 0.83\]

- Erasure coding:
  \[1 - \frac{1}{32} = 0.9765625\]
  \[1 - \frac{1}{31} = 0.99347826125\]
Ensuring Data Availability

Adversary → Light Node

“Is the Block Available?”

Request/sample few random chunks of the block

Adversary can hide a small portion

No coding:

Probability of failure using 2 random samples:

\[ 1 - \frac{1}{16} \approx 0.87 \]

Erasure coding:

Probability of failure using 2 random samples:

\[ 1 - \frac{1}{17} \approx 0.921 \]
Ensuring Data Availability

- Request/sample few random chunks of the block

Adversary → Light Node

"Is the Block Available?"
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

“Is the Block Available?”
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

Probability of failure using 2 random samples:

\[
1 - \frac{1}{16} = 0.87
\]

Erasure coding:

\[
1 - \frac{1}{32} = 0.9921
\]
Ensuring Data Availability

▶ Request/sample few random chunks of the block
▶ Adversary can hide a small portion

“Is the Block Available?”

Block → Small portion hidden
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

“Is the Block Available?”

<table>
<thead>
<tr>
<th>Block</th>
<th>Small portion hidden</th>
<th>Random chunks requested</th>
</tr>
</thead>
</table>

No coding:

\[
\text{Probability of failure using 2 random samples:} \quad 1 - \frac{1}{16} = 0.87
\]

Erasure coding:

\[
\text{Probability of failure using 2 random samples:} \quad 1 - \frac{1}{32} = 0.21
\]
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

Probability of failure using 2 random samples:

\[
1 - \left(1 - \frac{1}{16}\right)^2 = 0.87
\]

\[
1 - \left(1 - \frac{1}{32}\right)^2 = 0.21
\]
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

Probability of failure using 2 random samples:

\[
(1 - \frac{1}{16}) (1 - \frac{1}{15}) = 0.87
\]
Ensuring Data Availability

- Request/sample few random chunks of the block
- Adversary can hide a small portion

No coding:

Block → Small portion hidden • Random chunks requested

Probability of failure using 2 random samples:

\[
(1 - \frac{1}{16}) (1 - \frac{1}{15}) = 0.87
\]

Erasure coding:

Block → (32,16) MDS code
Ensuring Data Availability

▶ Request/sample few random chunks of the block
▶ Adversary can hide a small portion

No coding:

```
Block
```

```
Small portion hidden
Random chunks requested
```

```
Probability of failure using 2 random samples:

\[
(1 - \frac{1}{16}) (1 - \frac{1}{15}) = 0.87
\]
```

Erasure coding:

```
Block
```

```
(32,16) MDS code
```
Ensuring Data Availability

Request/sample few random chunks of the block
Adversary can hide a small portion

No coding:

Block → [Random chunks requested, Small portion hidden]

```
Probability of failure using 2 random samples:

\( 1 - \frac{1}{16} \) \( 1 - \frac{1}{15} \) = 0.87
```

Erasure coding:

Block → (32,16) MDS code

```
Probability of failure using 2 random samples:

\( 1 - \frac{17}{32} \) \( 1 - \frac{17}{31} \) = 0.21
```
Choice of Code Matters

Incorrect coding attack:
• Adversary sends incorrectly coded block to Full Nodes
• Honest Full nodes can detect and send incorrect coding proof
• Incorrect coding proof size: $O(s)$ (sparsity of parity check equations)

MDS codes: proof size = $O(b)$ (block size)

Decoding complexity

Undecodable ratio $\alpha$
• Probability of Light node failure using $s$ random samples = $(1 - \alpha)^s$
Choice of Code Matters

- Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size: $O(sparsity\ of\ parity\ check\ equations)$
  - MDS codes: proof size = $O(block\ size)$

- Decoding complexity
- Undecodable ratio $\alpha$

Probability of Light node failure using $s$ random samples = $(1 - \alpha)^s$
Choice of Code Matters

Incorrect coding attack:
- Adversary sends incorrectly coded block to Full Nodes

Incorrectly generated parity
Choice of Code Matters

Incorrect coding attack:
- Adversary sends incorrectly coded block to Full Nodes
- Honest Full nodes can detect and send incorrect coding proof
- Incorrect coding proof size: $\mathcal{O}(\text{sparsity of parity check equations})$

Diagram:
- Coding
- Incorrectly generated parity
Choice of Code Matters

► Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size: $\mathcal{O}(\text{sparsity of parity check equations})$
  - MDS codes: proof size $= \mathcal{O}(\text{block size})$
Choice of Code Matters

- Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size: $\mathcal{O}(\text{sparsity of parity check equations})$
  - MDS codes: proof size $= \mathcal{O}(\text{block size})$

- Decoding complexity
Choice of Code Matters

- Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size: $\mathcal{O}(\text{sparsity of parity check equations})$
  - MDS codes: proof size = $\mathcal{O}(\text{block size})$

- Decoding complexity
- Undecodable ratio $\alpha$
Choice of Code Matters

- Incorrect coding attack:
  - Adversary sends incorrectly coded block to Full Nodes
  - Honest Full nodes can detect and send incorrect coding proof
  - Incorrect coding proof size: $O($sparsity of parity check equations$)$
  - MDS codes: proof size = $O($block size$)$

- Decoding complexity

- Undecodable ratio $\alpha$
  - Probability of Light node failure using $s$ random samples = $(1 - \alpha)^s$
LDPC Codes: A Strong Contender

LDPC codes:
- Characterized by a sparse parity check matrix
LDPC Codes: A Strong Contender

LDPC codes:
- Characterized by a sparse parity check matrix
- Tanner Graph

*circles: variable nodes (VNs)*
*squares: check nodes (CNs)*
**LDPC Codes: A Strong Contender**

LDPC codes:
- Characterized by a sparse parity check matrix
- Tanner Graph

LDPC codes have been shown to be suitable for this application [Yu’ 19]
LDPC Codes: A Strong Contender

LDPC codes:
- Characterized by a sparse parity check matrix
- Tanner Graph

LDPC codes have been shown to be suitable for this application [Yu’ 19]
- Small incorrect coding proof size due to small check node degree
LDPC Codes: A Strong Contender

LDPC codes:

- Characterized by a sparse parity check matrix

- Tanner Graph

LDPC codes have been shown to be suitable for this application [Yu’ 19]

- Small incorrect coding proof size due to small check node degree
- Linear decoding in terms of the block size using peeling decoder
LDPC Codes: A Strong Contender

LDPC codes:
- Characterized by a sparse parity check matrix
- Tanner Graph

LDPC codes have been shown to be suitable for this application [Yu’ 19]
- Small incorrect coding proof size due to small check node degree
- Linear decoding in terms of the block size using peeling decoder
- What about the undecodable ratio?
Challenge with LDPC Codes: Small Stopping Sets

- Substructure in the Tanner Graph

\[
1 - \frac{3}{32} = 0.81
\]

Our work: Design of specialized LDPC codes with a coupled sampling strategy to achieve a significantly lower probability of failure.
Challenge with LDPC Codes: Small Stopping Sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block → No fraud proof

![Diagram of Tanner Graph and LDPC code]

Our work: Design of specialized LDPC codes with a coupled sampling strategy to achieve a significantly lower probability of failure:

\[
1 - \frac{3}{32} = 0.81
\]
Challenge with LDPC Codes: Small Stopping Sets

▶ Substructure in the Tanner Graph
▶ If hidden, prevents peeling decoder from decoding the block → No fraud proof

\[
\left(1 - \frac{3}{32}\right) \left(1 - \frac{3}{31}\right) = 0.81
\]

Our work: Design of specialized LDPC codes with a coupled sampling strategy to achieve a significantly lower probability of failure.
Challenge with LDPC Codes: Small Stopping Sets

- Substructure in the Tanner Graph
- If hidden, prevents peeling decoder from decoding the block → No fraud proof

\[
\left(1 - \frac{3}{32}\right) \left(1 - \frac{3}{31}\right) = 0.81
\]

Our work: Design of specialized LDPC codes with a coupled sampling strategy to achieve a significantly lower probability of failure.
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.

- Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.
- Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
- Stronger adversaries have been considered in full paper [Mitra ’21]
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.

Bob Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
Bob Stronger adversaries have been considered in full paper [Mitra ’21]

Lemma

Of all stopping sets (SSs) of size \( \mu \), when an adversary randomly hides one of them, and light nodes sample all VNs in the set \( \mathcal{L} \), then
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.

- Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
- Stronger adversaries have been considered in full paper [Mitra ’21]

**Lemma**

Of all stopping sets (SSs) of size $\mu$, when an adversary randomly hides one of them, and light nodes sample all VNs in the set $\mathcal{L}$, then

$\text{Probability of failure} = 1 - \frac{\text{fraction of SSs of size } \mu \text{ touched by } \mathcal{L}}{\text{number of SSs}}$
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.

- Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
- Stronger adversaries have been considered in full paper [Mitra ’21]

Lemma

Of all stopping sets (SSs) of size $\mu$, when an adversary randomly hides one of them, and light nodes sample all VNs in the set $\mathcal{L}$, then

\[
\text{Probability of failure} = 1 - \text{fraction of SSs of size } \mu \text{ touched by } \mathcal{L}
\]

- Selecting a set $\mathcal{L}$ of VNs which touches large no. of SSs
Motivation: Not all VNs are equal

In this presentation, we consider an adversary that randomly hides a stopping set of a particular size.

- Relevant in IoT blockchains [Wang ’19], Proof-of-Stake blockchains [Daian ’19]
- Stronger adversaries have been considered in full paper [Mitra ’21]

Lemma

Of all stopping sets (SSs) of size \( \mu \), when an adversary randomly hides one of them, and light nodes sample all VNs in the set \( \mathcal{L} \), then

\[
\text{Probability of failure} = 1 - \frac{\text{fraction of SSs of size } \mu \text{ touched by } \mathcal{L}}{}
\]

- Selecting a set \( \mathcal{L} \) of VNs which touches large no. of SSs
  \[\rightarrow \text{Prob. of failure} \downarrow\]
Concentrated Stopping Set Design

Code Design Idea:

- Concentrate stopping sets to a small section of VNs
Concentrated Stopping Set Design

Code Design Idea:
- Concentrate stopping sets to a small section of VNs
Concentrated Stopping Set Design

Code Design Idea:

- Concentrate stopping sets to a small section of VNs
- Greedily Sample this small section of VNs

Greedily Sampled VNs

Concentrated Region

SS distribution

Fraction of SSs touched

Concentrated VNs
How to Concentrate Stopping Sets?

▶ When there are no degree 1 VN s, stopping sets are either cycles or interconnection of cycles [Tian '03]
How to Concentrate Stopping Sets?

- When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian ’03]
- Concentrating cycles $\implies$ Concentrating stopping sets
How to Concentrate Stopping Sets?

- When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian ’03]

- Concentrating cycles $\Rightarrow$ Concentrating stopping sets
  - Directly concentrating stopping sets during code construction incurs huge complexity
How to Concentrate Stopping Sets?

- When there are no degree 1 VN nodes, stopping sets are either cycles or interconnection of cycles [Tian '03]
- Concentrating cycles $\implies$ Concentrating stopping sets
  - Directly concentrating stopping sets during code construction incurs huge complexity

- How to design codes with concentrated cycles?
How to Concentrate Stopping Sets?

- When there are no degree 1 VNs, stopping sets are either cycles or interconnection of cycles [Tian '03]
- Concentrating cycles $\implies$ Concentrating stopping sets
  - Directly concentrating stopping sets during code construction incurs huge complexity

- How to design codes with concentrated cycles?
  We do so by modifying the well-known Progressive Edge Growth (PEG) algorithm
PEG Algorithm

- Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]
PEG Algorithm

- Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]

For each VN $v_j$
Expand Tanner Graph in a BFS fashion
PEG Algorithm

Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

For each VN $v_j$
Expand Tanner Graph in a BFS fashion
If $\exists$ CNs not connected to $v_j$

No new CNs

CNs not connected to $v_j$
PEG Algorithm

Constructs a Tanner Graph in an edge by edge manner [Xiao '05]

For each VN $v_j$
Expand Tanner Graph in a BFS fashion
If $\exists$ CNs not connected to $v_j$
  - Select a CN with min degree not connected to $v_j$
PEG Algorithm

Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]

For each VN $v_j$
Expand Tanner Graph in a BFS fashion
If $\exists$ CNs not connected to $v_j$
• Select a CN with min degree not connected to $v_j$
Else

All CNs exhausted
PEG Algorithm

- Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]

**For each VN $v_j$**
- Expand Tanner Graph in a BFS fashion
  - **If** $\exists$ CNs not connected to $v_j$
    - Select a CN with min degree not connected to $v_j$
  - **Else**
    - Find CNs most distant to $v_j$
    - Select one with minimum degree

All CNs exhausted
PEG Algorithm

- Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]

**For each VN** $v_j$

Expand Tanner Graph in a BFS fashion

- **If** $\exists$ CNs not connected to $v_j$
  - Select a CN with min degree not connected to $v_j$

- **Else**
  - Find CNs most distant to $v_j$
  - Select one with minimum degree

*New cycles created*

All CNs exhausted
**PEG Algorithm**

Constructs a Tanner Graph in an edge by edge manner [Xiao ’05]

For each VN $v_j$

Expand Tanner Graph in a BFS fashion

If $\exists$ CNs not connected to $v_j$

- Select a CN with min degree not connected to $v_j$

Else

- Find CNs most distant to $v_j$
- Select one with minimum degree

*New cycles created*

We modify the CN selection criteria in green to concentrate cycles

All CNs exhausted
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $\mathcal{H}(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $\mathcal{H}(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$

- Uniform distributions have high entropy

![Graph showing high entropy distribution](image)
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $H(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$

- Uniform distributions have high entropy
- Concentrated distributions have low entropy
Using Entropy to Concentrate Cycles

For distribution \( p = (p_1, p_2, \ldots, p_n) \), Entropy \( \mathcal{H}(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \)

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

We want the cycle distributions to be concentrated
Using Entropy to Concentrate Cycles

For distribution \( p = (p_1, p_2, \ldots, p_n) \), Entropy \( H(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \)

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

High Entropy

Low Entropy

We want the cycle distributions to be concentrated
→ Select CNs such that the entropy of the cycle distribution is minimized
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $\mathcal{H}(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

EC (Entropy Constrained)-PEG Algorithm

For each VN $v_j$

Expand Tanner Graph in a BFS fashion

If $\exists$ CNs not connected to $v_j$

- select a CN with min degree not connected to $v_j$

Else New cycles created

We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $H(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

EC (Entropy Constrained)-PEG Algorithm

For each VN $v_j$

Expand Tanner Graph in a BFS fashion

If $\exists$ CNs not connected to $v_j$

- select a CN with min degree not connected to $v_j$

Else New cycles created

- Find CNs most distant to $v_j$

We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized
Using Entropy to Concentrate Cycles

For distribution $p = (p_1, p_2, \ldots, p_n)$, Entropy $\mathcal{H}(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

EC (Entropy Constrained)-PEG Algorithm

For each VN $v_j$
- Expand Tanner Graph in a BFS fashion
  - If $\exists$ CNs not connected to $v_j$
    - select a CN with min degree not connected to $v_j$
  - Else New cycles created
    - Find CNs most distant to $v_j$
    - Select CN that results in minimum entropy of resultant cycle distribution

We want the cycle distributions to be concentrated
→ Select CNs such that the entropy of the cycle distribution is minimized
Using Entropy to Concentrate Cycles

For distribution \( p = (p_1, p_2, \ldots, p_n) \), Entropy \( H(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \):

- Uniform distributions have high entropy
- Concentrated distributions have low entropy

**EC (Entropy Constrained)-PEG Algorithm**

For each VN \( v_j \):

Expand Tanner Graph in a BFS fashion

If \( \exists \) CNs not connected to \( v_j \):

- select a CN with min degree not connected to \( v_j \)

Else **New cycles created**:

- Find CNs most distant to \( v_j \)
- Select CN that results in minimum entropy of resultant cycle distribution
- Update cycle distribution

We want the cycle distributions to be concentrated

→ Select CNs such that the entropy of the cycle distribution is minimized
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN.
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[ \text{VNs} \ (v_1, v_2, \ldots, v_n) \]
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[ \text{VNs} \ (v_1, v_2, \ldots, v_n) \]

\[ \lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8 \]
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

VNs \((v_1, v_2, \ldots, v_n)\)

\(\lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8\)
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

$\lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8$
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN.

\( VNs (v_1, v_2, \ldots, v_n) \)

\( \lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8 \)
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[
\lambda_i(g) := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8
\]

VNs \((v_1, v_2, \ldots, v_n)\)
EC-PEG Algorithm

- Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[
\text{VNs } \{v_1, v_2, \ldots, v_n\}
\]

\(\lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8\)

\[
\begin{align*}
\lambda_1^{(6)} &= \lambda_1^{(6)} + 1 \\
\lambda_2^{(6)} &= \lambda_2^{(6)} + 1 \\
\lambda_6^{(6)} &= \lambda_6^{(6)} + 1
\end{align*}
\]
EC-PEG Algorithm

Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[ \lambda_i^{(g)} := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8 \]

\[
\begin{align*}
\lambda_1^{(6)} &= \lambda_1^{(6)} + 1 \\
\lambda_2^{(6)} &= \lambda_2^{(6)} + 1 \\
\lambda_7^{(6)} &= \lambda_7^{(6)} + 1
\end{align*}
\]
Whenever a new edge, that creates cycles, is added to the Tanner Graph, we update the cycle counts of each VN

\[
\lambda_i(g) := \text{No. of cycles of length } g \text{ that } v_i \text{ is a part of, } g = 4, 6, 8
\]

\[
\begin{align*}
\lambda_1^{(6)} &= \lambda_1^{(6)} + 1 \\
\lambda_3^{(6)} &= \lambda_3^{(6)} + 1 \\
\lambda_6^{(6)} &= \lambda_6^{(6)} + 1
\end{align*}
\]
EC-PEG Algorithm: CN Selection Procedure

Candidate CNs: $c_8$, $c_9$, $c_{10}$
EC-PEG Algorithm: CN Selection Procedure

Candidate CNs: \( c_8, c_9, c_{10} \)

- For each CN candidate, calculate the resultant VN cycle counts

\[ (\lambda(4)_1, \ldots, \lambda(4)_n), (\lambda(6)_1, \ldots, \lambda(6)_n), (\lambda(8)_1, \ldots, \lambda(8)_n) \]
EC-PEG Algorithm: CN Selection Procedure

Candidate CNs: $c_8, c_9, c_{10}$

- For each CN candidate, calculate the resultant VN cycle counts
- $(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})$
EC-PEG Algorithm: CN Selection Procedure

Candidate CNs: \(c_8, c_9, c_{10}\)

- For each CN candidate, calculate the resultant VN cycle counts
  - \((\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})\)
  - \((\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})\)
EC-PEG Algorithm: CN Selection Procedure

Candidate CNs: $c_8$, $c_9$, $c_{10}$

- For each CN candidate, calculate the resultant VN cycle counts
  - $(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})$
  - $(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})$
  - $(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})$
EC-PEG Algorithm: CN Selection Procedure

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]
EC-PEG Algorithm: CN Selection Procedure

\[ (\lambda^{(4)}_1, \ldots, \lambda^{(4)}_n), (\lambda^{(6)}_1, \ldots, \lambda^{(6)}_n), (\lambda^{(8)}_1, \ldots, \lambda^{(8)}_n) \]

\[ (\lambda^{(4)}_1, \ldots, \lambda^{(4)}_n), (\lambda^{(6)}_1, \ldots, \lambda^{(6)}_n), (\lambda^{(8)}_1, \ldots, \lambda^{(8)}_n) \]

\[ (\lambda^{(4)}_1, \ldots, \lambda^{(4)}_n), (\lambda^{(6)}_1, \ldots, \lambda^{(6)}_n), (\lambda^{(8)}_1, \ldots, \lambda^{(8)}_n) \]

\[ (\lambda^{(g)}_1, \ldots, \lambda^{(g)}_n) \]

\underbrace{\text{cycle counts}}
EC-PEG Algorithm: CN Selection Procedure

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(g)}, \ldots, \lambda_n^{(g)}) \rightarrow \left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}, \ldots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}} \right) := \alpha^{(g)} \]

\[ \text{cycle counts} \quad \rightarrow \quad \text{normalized counts} \]
EC-PEG Algorithm: CN Selection Procedure

\[
(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})
\]

\[
(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})
\]

\[
(\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})
\]

\[
\left(\lambda_1^{(g)} , \ldots, \lambda_n^{(g)}\right) \overset{\text{cycle counts}}{\rightarrow} \left(\frac{\lambda_1^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}, \ldots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}\right) := \alpha^{(g)} \rightarrow \mathcal{H}\left(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}\right) \overset{\text{entropy of combined counts}}{\rightarrow}
\]
EC-PEG Algorithm: CN Selection Procedure

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \rightarrow H(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \]

\[ (\lambda_1^{(g)}, \ldots, \lambda_n^{(g)}) \rightarrow \left( \frac{\lambda_1^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}, \ldots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}} \right) := \alpha^{(g)} \rightarrow H(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}) \]

*cycle counts*

*normalized counts*

*entropy of combined counts*
EC-PEG Algorithm: CN Selection Procedure

- \((\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})\)
- \((\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)}) \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})\)
- \((\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}), (\lambda_1^{(6)}, \ldots, \lambda_n^{(6)}), (\lambda_1^{(8)}, \ldots, \lambda_n^{(8)})\)

\[
\begin{align*}
(\lambda_1^{(g)}, \ldots, \lambda_n^{(g)}) &\rightarrow \left(\frac{\lambda_1^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}, \ldots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}} \right) := \alpha^{(g)} \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})
\end{align*}
\]

Cycle counts $$\rightarrow$$ Normalized counts $$\rightarrow$$ Entropy of combined counts
EC-PEG Algorithm: CN Selection Procedure

\[
\begin{align*}
\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}, \lambda_1^{(6)}, \ldots, \lambda_n^{(6)}, \lambda_1^{(8)}, \ldots, \lambda_n^{(8)} & \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}) \\
\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}, \lambda_1^{(6)}, \ldots, \lambda_n^{(6)}, \lambda_1^{(8)}, \ldots, \lambda_n^{(8)} & \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}) \\
\lambda_1^{(4)}, \ldots, \lambda_n^{(4)}, \lambda_1^{(6)}, \ldots, \lambda_n^{(6)}, \lambda_1^{(8)}, \ldots, \lambda_n^{(8)} & \rightarrow \mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})
\end{align*}
\]

\[
\left(\lambda_1^{(g)}, \ldots, \lambda_n^{(g)}\right) \rightarrow \left(\frac{\lambda_1^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}, \ldots, \frac{\lambda_n^{(g)}}{\sum_{i=1}^{n} \lambda_i^{(g)}}\right) := \alpha^{(g)} \rightarrow \mathcal{H}\left(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3}\right)
\]

- Cycle counts
- Normalized counts
- Entropy of combined counts
EC-PEG Algorithm: CN Selection Procedure

\[ \lambda(4)^1, \ldots, \lambda(4)^n, \lambda(6)^1, \ldots, \lambda(6)^n, \lambda(8)^1, \ldots, \lambda(8)^n \rightarrow H(\alpha(4) + \alpha(6) + \alpha(8)) \]

\[ \lambda(4)^1, \ldots, \lambda(4)^n, \lambda(6)^1, \ldots, \lambda(6)^n, \lambda(8)^1, \ldots, \lambda(8)^n \rightarrow H(\alpha(4) + \alpha(6) + \alpha(8)) \]

\[ \lambda(4)^1, \ldots, \lambda(4)^n, \lambda(6)^1, \ldots, \lambda(6)^n, \lambda(8)^1, \ldots, \lambda(8)^n \rightarrow H(\alpha(4) + \alpha(6) + \alpha(8)) \]

CN selection procedure:
EC-PEG Algorithm: CN Selection Procedure

Select CN that results in minimum $\mathcal{H}(\frac{\alpha^{(4)} + \alpha^{(6)} + \alpha^{(8)}}{3})$
EC-PEG Algorithm: CN Selection Procedure

Note:

- Minimizing the entropy of joint cycle counts ensures that all cycle distributions are concentrated towards the same set of VNs
Our sampling strategy greedily samples VNs that are part of a large number of cycles.

\[ g = \text{smallest cycle length in Tanner Graph } G \]

**While** sample set size < \( s \)

- \( v \) = VN that is part of largest no. of cycles of length \( g \) in \( G \)
- sample set = sample set \( \cup \) \( v \)
- remove \( v \) and all incident edges from \( G \)
Sampling Strategy

- Our sampling strategy greedily samples VNs that are part of a large number of cycles

\[ g = \text{smallest cycle length in Tanner Graph } G \]
\[ \textbf{While} \quad \text{sample set size} < s \]
  - \( v = \text{VN that is part of largest no. of cycles of length } g \text{ in } G \)
  - \( \text{sample set} = \text{sample set} \cup v \)
  - \( \text{remove } v \text{ and all incident edges from } G \)
\[ \textbf{If} \quad \# \text{cycles of length } g \text{ in } G \]
  - \( g = g + 2 \)
Simulation Results

- Code parameters: Code length = 100, VN degree = 4, Rate = $\frac{1}{2}$, girth = 6.
Simulation Results

- Code parameters: Code length = 100, VN degree = 4, Rate = $\frac{1}{2}$, girth = 6.

- VN indices arranged in decreasing order of cycle 6 fractions
Simulation Results

- Code parameters: Code length = 100, VN degree = 4, Rate = \( \frac{1}{2} \), girth = 6.

- VN indices arranged in decreasing order of cycle 6 fractions
- Cycle 6 and cycle 8 concentrated towards same set of VNs
Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs
Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs

- SSs of size 11
  - VN indices arranged in decreasing order of cycle 6 fractions

- SSs of size 12

![Graph showing the fraction of SSs touched by different VNs for Original PEG and EC-PEG methods.](image)
Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs

▶ VN indices arranged in decreasing order of cycle 6 fractions
Simulation Results

Fraction of SSs of size 11, 12 touched by different VNs

- VN indices arranged in decreasing order of cycle 6 fractions
- SSs are concentrated towards the same set of VNs as the cycles
Simulation Results
Probability of failure for a stopping set of size $\mu$
Simulation Results
Probability of failure for a stopping set of size $\mu$

RS: Random Sampling

Note that the probability of failure depends on the fraction of stopping sets touched (by greedy sampling) and not the actual number.

Mitra, Tauz, Dolecek (UCLA)
Simulation Results
Probability of failure for a stopping set of size $\mu$

- RS: Random Sampling

Note that the probability of failure depends on the fraction of stopping sets touched (by greedy sampling) and not the actual number.

Mitra, Tauz, Dolecek (UCLA)
Simulation Results

Probability of failure for a stopping set of size $\mu$

RS: Random Sampling
GS: Greedy Sampling

Note that the probability of failure depends on the fraction of stopping sets touched (by greedy sampling) and not the actual number.

Mitra, Tauz, Dolecek (UCLA)
**Simulation Results**

Probability of failure for a stopping set of size $\mu$

RS: Random Sampling  
GS: Greedy Sampling

Note that the probability of failure depends on the fraction of stopping sets touched (by greedy sampling) and not the actual number.

---

Mitra, Tauz, Dolecek (UCLA)
Simulation Results

Probability of failure for a stopping set of size $\mu$

- Concentrated LDPC codes with Greedy sampling improve the probability of failure

RS: Random Sampling
GS: Greedy Sampling
Simulation Results

Probability of failure for a stopping set of size $\mu$

- Concentrated LDPC codes with Greedy sampling improve the probability of failure

\[\text{Note that the probability of failure depends on the fraction of stopping sets touched (by greedy sampling) and not the actual number.}\]
Incorrect Coding Proof Size

- Depends on the maximum check node degree

<table>
<thead>
<tr>
<th>Rate</th>
<th>Code length</th>
<th>VN degree</th>
<th>Ensemble [Yu ’19]</th>
<th>PEG</th>
<th>EC-PEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2}$</td>
<td>100</td>
<td>4</td>
<td>16</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>4</td>
<td>16</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>$\frac{1}{4}$</td>
<td>100</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Table: Maximum CN degree for different codes.
Incorrect Coding Proof Size

- Depends on the maximum check node degree

<table>
<thead>
<tr>
<th>Rate</th>
<th>Code length</th>
<th>VN degree</th>
<th>Ensemble [Yu ’19]</th>
<th>PEG</th>
<th>EC-PEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2}$</td>
<td>100</td>
<td>4</td>
<td>16</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>4</td>
<td>16</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>$\frac{1}{4}$</td>
<td>100</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Table: Maximum CN degree for different codes.

- Concentrated LDPC codes do not sacrifice on the incorrect coding proof size
Conclusion and Extensions

Summary:

- We provided a specialized code construction technique to concentrate stopping sets in LDPC codes

Extensions (Mitra '21):

- Considered stronger adversary models that can selectively pick a stopping set that has a lower probability of being sampled to hide instead of randomly
- Provided optimal sampling strategies and associated coupled LDPC code construction to improve the security against such strong adversaries for a given sample complexity
Conclusion and Extensions

Summary:

- We provided a specialized code construction technique to concentrate stopping sets in LDPC codes.
- Coupled with a greedy sampling strategy, concentrated LDPC codes reduce the probability of light node failure compared to earlier approaches.

Extensions (Mitra ‘21):
- Considered stronger adversary models that can selectively pick a stopping set that has a lower probability of being sampled to hide instead of randomly.
- Provided optimal sampling strategies and associated coupled LDPC code construction to improve the security against such strong adversaries for a given sample complexity.
Conclusion and Extensions

Summary:
- We provided a specialized code construction technique to concentrate stopping sets in LDPC codes
- Coupled with a greedy sampling strategy, concentrated LDPC codes reduce the probability of light node failure compared to earlier approaches

Extensions (Mitra '21):
- Considered stronger adversary models that can selectively pick a stopping set that has a lower probability of being sampled to hide instead of randomly
Conclusion and Extensions

▶ Summary:
- We provided a specialized code construction technique to concentrate stopping sets in LDPC codes
- Coupled with a greedy sampling strategy, concentrated LDPC codes reduce the probability of light node failure compared to earlier approaches

▶ Extensions (Mitra '21):
- Considered stronger adversary models that can selectively pick a stopping set that has a lower probability of being sampled to hide instead of randomly
- Provided optimal sampling strategies and associated coupled LDPC code construction to improve the security against such strong adversaries for a given sample complexity
References


References